Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring

Peter Waszecki[1], Philipp Mundhenk[2], Sebastian Steinhorst[4], Martin Lukasiewycz[2], Ramesh Karri[3], Samarjit Chakraborty[4]

[1] Singapore Institute of Technology, Email: peter.waszecki@singaporetech.edu.sg,
[2] TUM CREATE, Singapore,[3] NYU, United States,[4] TU Munich, Germany

***Abstract*—Due to the growing interconnectedness and complexity of in-vehicle networks, in addition to safety, *security* is becoming an increasingly important topic in the automotive domain. In this paper we study techniques for detecting security infringements in automotive Electrical and Electronic (E/E) architectures. Towards this we propose in-vehicle network traffic monitoring to detect increased transmission rates of manipulated message streams. Attacks causing timing violations can disrupt safety-critical functions and have severe consequences. To reduce costs and prevent single points of failure, our approach enables an automatic distribution of detection tasks among selected E/E architecture components, such as a subset of Electronic Control Units (ECUs). First, we analyze a concrete E/E system architecture to determine the communication parameters and properties necessary for detecting security attacks. These are then used for a parametrization of the corresponding detection algorithms and the distribution of attack detection tasks. We use a lightweight message monitoring method and optimize the placement of detection tasks to ensure a full-coverage of the E/E system architecture and a timely detection of an attack.**

***Index Terms*—Automotive security, distributed systems, attack detection, embedded systems**

I. Introduction

Instead of simply being modes of conveyance as in the past, today cars are also expected to entertain and inform passengers in a safe and protected environment. Further, they are supposed to ease the complexity of driving by providing different forms of assistance to the driver. To support these increasing demands, automotive E/E architectures have become highly complex with over 100 ECUs communicating via multiple automotive-specific buses and gateways. Moreover, emerging functionality like Car-to-Car (C2C) and Car-to-Infrastructure (C2I) communication, as well as infotainment and driver assistance systems have increased the number of vehicle components with communication interfaces to the outside world[1]. Such complex E/E and connected automotive architectures are increasingly vulnerable to malicious attacks [1] and recent demonstrations of attacks in production vehicles clearly illustrate the growing importance of this topic [2]. Considering security side-by-side with safety is crucial for the overall reliability of an automotive E/E architecture, since a vulnerable electronic component might undermine the passengers' safety to the same extent as a faulty one [3]. However, in the highly competitive automotive domain, the cost of security features often pose an obstacle to their adoption.

In this paper, we present a decentralized and non-intrusive automotive E/E attack monitoring approach that is effective while being cost-efficient. In message-oriented networks, message flooding and Denial of Service (DoS) attacks change the timing behavior in message streams whose transmission patterns are altered. Early detection of timing violations is crucial in vehicular networks, since an increase in the expected message rate can compromise safety-critical functions, for instance, by impeding the responsiveness in a brake-by-wire system. Automotive E/E architectures implement timing-critical control software where message periods and execution time bounds are usually prescribed, such that delays and worst case jitters can be calculated[2]. Since many methods to infiltrate and disrupt automotive in-vehicle networks manifest as altered packet transmission patterns, we propose detecting deviations in the timing specifications of individual message streams. Our approach does not analyze the message *content* but is focused on detecting manipulations in the *communication behavior*.

In contrast to classic DoS attacks, our method can also detect attacks where a service is not entirely disabled but, for instance, carried out incorrectly. It has been shown that some attacks increase the rate of a regular message stream and flood the bus using highest-priority Controller Area Network (CAN) identifiers, thereby restricting the availability of the bus [4]. This can seriously disrupt other traffic and associated functionality, for example, by suddenly turning off the headlights at night [5, 6]. A discussion of existing work on automotive security is presented in Section VI.

The approach presented here considers the guidance on vehicle security that was recently proposed by the National Highway Traffic Safety Administration (NHTSA) [7]. Here, the recommended Best Practices provide a flexible and pragmatic way to ensure security in automotive cyber-physical systems [8]. By supporting anomaly detection in vehicle operations, our proposed detection of message-based attacks

This work was financially supported in part by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme. With the support of the Technische Universität München - Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n 291763.

[1]Most of this data is transmitted via radio-based communication links.

[2]Generic computer networks are based on dynamic and unpredictable packet transmission schemes where timing violation attacks can only be detected when changes in the traffic are sufficiently large.

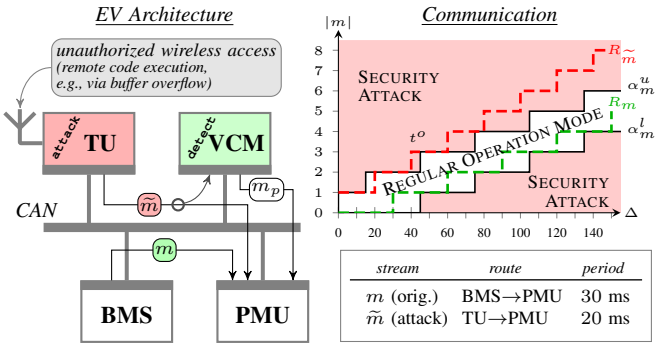

Fig. 1: The telematics unit (TU) of an automotive E/E architecture is compromised and used to manipulate the communication between the battery management system (BMS) and the power management unit (PMU). The attack can be detected by the vehicle control module (VCM) by testing the deviation of the manipulated stream $\widetilde{m}$ from the original stream m.

falls into one of the seven key security functions of NHTSA's Best Practices.

Motivating example. Consider the example in Figure 1 that demonstrates the manipulation of particular in-vehicle traffic patterns in an Electric Vehicle (EV) to disable the car's power supply. The figure depicts a section of an E/E architecture as used in a best-selling commercial EV [9]. It consists of 4 ECUs connected to a CAN bus: a telematics unit (TU) providing a wireless connection for remote services, a vehicle control module (VCM) evaluating, among others, ECU and sensor signals, a Battery Management System (BMS) controlling the Li-Ion cells and a power management unit (PMU). In EVs, to avoid a potentially harmful battery depletion, the transmission of energy to the motor or other electrical devices is initiated by a torque request (e.g., by the PMU) to the BMS and has to be accepted or rejected with a response message (message stream m from BMS to PMU). Moreover, to ensure functional safety, a BMS usually contains a hard-wired shut-off switch which will cut the electrical power supply, for instance, before current or voltage levels can fall below dangerous thresholds [10]. Let us assume, that an attacker intends to bring the vehicle to an immediate halt. An intruder can gain wireless access to the automotive architecture, e.g., via the TU interface or a Bluetooth adapter for the on-board diagnostics port (OBD-II port) [11]. After taking control of the TU, the attacker manipulates its code to transmit a high frequency stream of responses accepting the torque request (message stream $\widetilde{m}$), such that they dominate the original response message stream m which is possibly rejecting the torque request. As it is receiving positive responses to its torque request at a high rate, the PMU continues to draw current from the battery beyond the designated threshold which will trigger the shut-down switch. Disconnected from its power source, the car will abruptly slow down, lose its brake booster or power steering and come to an unexpected halt.

The proposed attack detection approach uses the knowledge of particular system components and information about essential communication parameters, such as message periods and maximum jitters. These parameters are specified during system design and can be stored as *message arrival curves* α_m. The curves are compact representations of the upper (α_m^u) and lower (α_m^l) bounds for the number of message stream events (i.e., messages) within an arbitrary time interval. As arrival curves are easily adaptable and can be parametrized for many different message stream patterns, they can help design efficient and distributed attack detection tasks. We are able to monitor the actual message streams and detect if and when the corresponding message count violates a predefined upper arrival curve α_m^u [3].

In our example, the VCM can detect a transgression of the upper arrival curve at time $t^o = 40$ ms, as illustrated in the graph on the right side of Figure 1. After detecting the altered message stream and identifying it as torque response via the CAN-ID, the VCM transmits a high-priority message m_p to the PMU informing it about the malicious responses from the BMS. The countermeasures could then initiate a safe and controlled reduction of the torque request and warn the driver about a serious malfunction asking him or her to stop the car. It is important that these countermeasures are initiated as early as possible. Moreover, should the attacker disrupt the corresponding detection functionality on the VCM, our decentralized redundancy-aware approach would be able to maintain the overall detection capability by using several distributed detection tasks for each message stream.

While there may exist other solutions to prevent the described security attack (e.g., separate buses and firewalls), the motivating example shall demonstrate our general approach of detecting timing violations and emphasize its non-intrusive, lightweight and cost-efficient aspects. We selected a realistic architecture in current electric vehicles making the presented method suitable and relevant for the automotive industry. Overall, the proposed attack detection is applicable to attacks which alter predefined message times, regardless of the entry point of the potential malicious attack program.

Contributions of the paper. The main goal of this work is to provide an efficient and reliable attack detection framework. First, a fast detection algorithm is formulated which can be efficiently implemented on each considered monitoring resource, such as an ECU, preferably close to the communication controller in order to minimize latencies. This algorithm is used to detect message streams which violate their predefined communication parameters. The message streams are represented by specific arrival curves which efficiently describe the earliest and latest times a message should be received and, thus, enable a timely detection of malicious attacks on the system communication. Second, the algorithm is implemented in detection tasks which can be automatically distributed to the monitoring resources while allowing the system designer to manually adjust necessary parameters such as level of redundancy and components utilization. In summary, we propose:

1) A real-time traffic conformity check to detect message-based attacks in automotive E/E architectures.
2) An optimization-based method to efficiently distribute these conformity checks among E/E monitoring resources considering redundancy and utilization.

Our approach utilizes the existing communication and the detection tasks are lightweight to ensure a low computational

[3]Message counts are depicted at runtime and do not represent arrival curves for arbitrary time intervals.

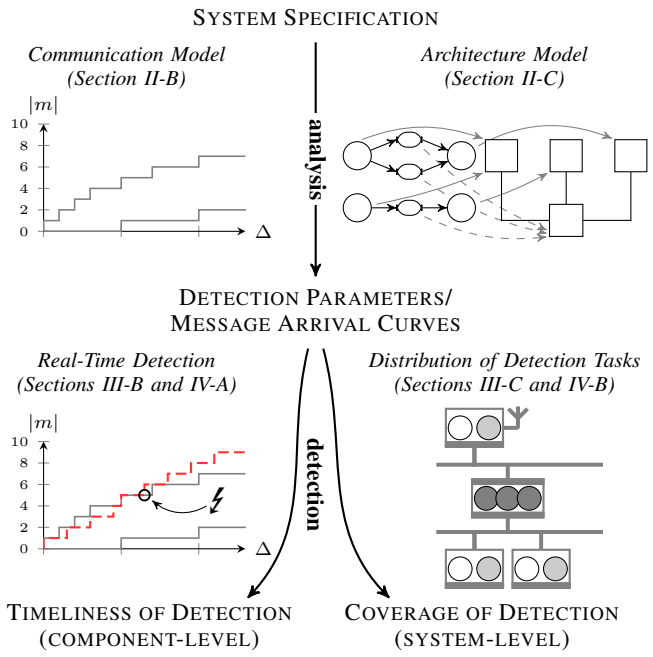

Fig. 2: The proposed automotive security framework is built up as follows: first, the system specification is analyzed to obtain the necessary message arrival curves and detection parameters. These are used to design the real-time attack detection and the distribution of detection tasks.

and timing overhead. Distributing the detection tasks removes the single point of failure. The proposed distributed detection method includes the entire system communication and allows configurations that consider redundancy and individual message streams. By contrast, while explicit security solutions such as firewalls might, in principle, be used for detecting timing violation as well, their implementation would involve considerable costs and changes to the in-vehicle network, for instance, by adding dedicated hardware and using separate buses. Additionally, a firewall would increase the risks associated with a single point of failure.

Finally, the focus of our paper is on the *detection* process for a secure automotive E/E architecture. The potential subsequent countermeasures are not part of this work. Furthermore, the proposed work does not claim sufficiency in terms of a universal security strategy. While the presented approach cannot guarantee to detect all types of security infringements, the low implementation effort and the piggybacking on the required E/E detection functionality offers added security orthogonal to other approaches.

Outline. The remainder of this paper is structured as follows. Section II presents the attack detection framework and provides a description of the communication and architecture models underlying the proposed approach. The detection method is introduced in Section III and the detection methodology is presented in Section IV. The limitations of our approach are discussed in detailed in Section III-A. Section V contains the experimental results which evaluate our approach based on synthetic test cases and a case study. Finally, the related work as well as the conclusion and future work are presented in Section VI and Section VII, respectively.

II. Framework and System Model

In this section, we present an overview of the security framework before introducing the communication and architecture models.

A. Security Framework

Figure 2 depicts the general structure of the proposed security framework. The first stage (upper part of the figure) illustrates the *system analysis* while the second stage (lower part of the figure) represents the *attack detection*. In the first stage, a system specification is analyzed in terms of its communication patterns and parameters, its applications and the architecture structure in order to obtain the necessary information for the attack detection, such as message arrival curves. In the second stage, message arrival curves are used to configure the real-time detection functions which are implemented on the monitoring platforms in a distributed manner. Additionally, the system architecture is considered when distributing the detection tasks using parameters such as redundancy levels and utilization and including the availability of E/E resources with monitoring capability. The framework will be evaluated with respect to the *timeliness* of detection and the *coverage* of the E/E system architecture which are used as two important metrics in safety-critical systems.

B. Communication Model

Although attack detection aims at the entire in-vehicle network independent of the particular communication protocol, it is reasonable to focus on an event-triggered buses such as CAN. It is by far the most prevalent automotive bus often used for safety-critical applications and CAN-based communication has the highest risk of being exploited [12]. As a consequence, to model the network traffic, we adopt arrival curves for general event-based systems [13]. These arrival curves are used to define the upper and lower bounds for the number of observed message streams on a bus during a specified time interval.

Message streams. A message stream m is generally described by its *event trace* function $\mu_m(n)$ where the n-th message arrives at time t_{n-1}, as defined in Equation (1).

$\forall t_n \in \mathbb{R}, n \in \mathbb{N}:$

$$\mu_m(n) = t_{n-1}, \quad \text{with } n \in \{1, ..., n_{\max}\} \tag{1}$$

For example, Figure 3 shows a segment of a message stream m where six message events arrive at times t_0 through t_5. Additionally, nine events of a compromised message stream $\widetilde{m}$ are depicted. For a message stream m, the *message count* $R_m[t_a, t_b)$ returns the number of messages arriving at a communication resource in the interval $[t_a, t_b)$, where t_a is included and t_b is excluded from the half-closed interval. This is formally defined in Equation (2a) with $R_m(t_a)$ and $R_m(t_b)$ determined by the maximum number of events of the trace function up to the time t_a and t_b, respectively (Equation (2b)).

$\forall t_a, t_b \in \mathbb{R}^+, t_a < t_b, n \in \mathbb{N}:$

$$R_m[t_a, t_b) = R_m(t_b) - R_m(t_a) \tag{2a}$$

$$R_m(t_{a/b}) = \max\left\{n \mid \mu_m(n) < t_{a/b}\right\} \tag{2b}$$

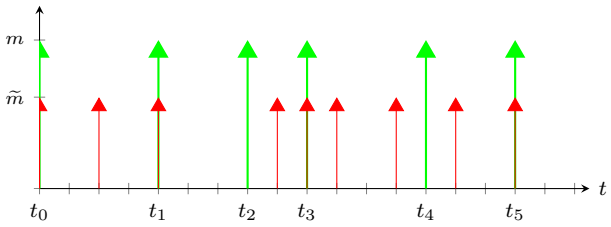

Fig. 3: Example for two event traces. Six green event arrivals (long arrows) represent an excerpt of a regular message stream m and nine red event arrivals (short arrows) represent a compromised message stream $\widetilde{m}$.

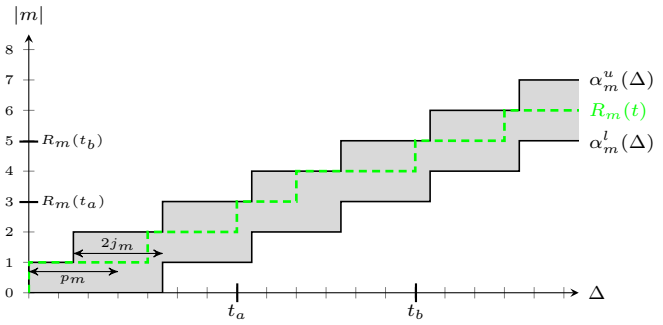

Fig. 4: The graph depicts the Regular Operation Mode on a communication resource (shaded area). $R_m(t)$ represents the message count for an example message stream which does not violate the bounds of a periodic with jitter communication defined by α^l_m and α^u_m. The adherence to Equations (3a) and (3b) is shown with the example of two message counts $R_m(t_a)$ and $R_m(t_b)$.

Arrival curves. For a message stream which is not periodic but is periodic with jitter, the message count can differ for the same interval length $[s, t)$ but different start times s. To detect communication inconsistencies, we are interested in determining the highest and lowest message count of m for each possible time interval. This can be facilitated by the aforementioned arrival curves. The minimum and maximum number of messages arriving in an arbitrary time interval Δ is bound by a lower arrival curve $\alpha^l_m(\Delta)$ and an upper arrival curve $\alpha^u_m(\Delta)$, respectively, both represented by *step functions*. Given the message count on a bus which lasts for an interval of length Δ, the lower and upper arrival curves must satisfy Equations (3a) and (3b), respectively.

$\forall t \in \mathbb{R}^+ :$

$$\alpha^l_m(\Delta) = \min_{t \geq 0}\{R_m(t+\Delta) - R_m(t)\} \tag{3a}$$

$$\alpha^u_m(\Delta) = \max_{t \geq 0}\{R_m(t+\Delta) - R_m(t)\} \tag{3b}$$

An example pair of arrival curves is shown in Figure 4. The graph depicts the number of messages in a stream m (*y-axis*) which can be observed within any arbitrary time interval Δ (*x-axis*). With the upper and lower arrival curves at its edges, the shaded area in the graph is the region of the Regular Operation Mode. Here, $R_m(t)$ is one of the infinite number of possible regular message streams. In this case it is representing the message arrivals of the stream m in Figure 3, where the observation begins at t_0 and t_a and t_b correspond to t_2 and t_4, respectively. It is important to understand, that $R_m(t)$ does not represent an arrival curve for an arbitrary time interval Δ but rather the message count of a particular message stream for a specific observation time. Hence, $R_m(t)$ does not refer to the x-axis of time intervals Δ but to an x-axis reflecting the absolute runtime t. In the graph, a separate axis for the latter one is omitted for the purpose of clarity.

Arrival curves determination. For message-based networks that serve mostly non-periodic communication, arrival curves can be determined from measurement or simulation. For example a window of length Δ can be slid over a traced message stream, recording the minimum and maximum message numbers. Repeating this for different window lengths allows the construction of the arrival curves. Some general strategies have been presented in [14]. However, the measurement and simulation approaches might not be sufficient as they still can miss the worst-case arrival times of messages.

The strict real-time constraints necessary to guarantee safety-critical functionality in automotive E/E architectures make the message transmission predictable. This allows for a reliable determination of the worst case timing parameters and, hence, the construction of reliable arrival curves at design-time. However, since during system design the arrival curves are initially calculated based on the task release times of the transmitting resources, the effects of existing bus traffic must be considered to assure the correct detection timing. More precisely, it is necessary to add delays to the nominal intervals of the arrival curves with respect to a possible current bus occupation and higher priority messages. For instance, in [12] this has been done for the CAN bus.

Generally, for many automotive applications, message streams can be modeled as *periodic with jitter*. In an event-triggered architecture, the jitter in a periodic message stream is an indicator for the variation between the inter-completion or response times of its corresponding successive release tasks. It can be caused by task preemption, changes in execution times or other delays [15]. Moreover, particular bus protocols can encounter jitters caused by the variations of the frame length due to bit-stuffing [16]. This means that the arrival curves α^l_m and α^u_m are shifted by the maximum possible jitter j_m of a message stream m relative to a nominal period p_m. Equation (4) defines the upper and lower arrival curve for an event-triggered system based on a periodic with jitter message stream. A detailed derivation of these equations is presented in [17].

$\forall p_m, j_m \in \mathbb{R}^+, \Delta > 0 :$

$$\alpha^l_m(\Delta) = \left\lfloor \frac{\Delta - j_m}{p_m} \right\rfloor, \qquad \alpha^u_m(\Delta) = \left\lceil \frac{\Delta + j_m}{p_m} \right\rceil \tag{4}$$

C. Architecture Model

The attack detection framework employs a graph-based system specification where software functionality and hardware are modeled by process and architecture graphs, respectively. Figure 5 illustrates this with two applications, a_1 and a_2, implemented on a small system architecture. Here, the process

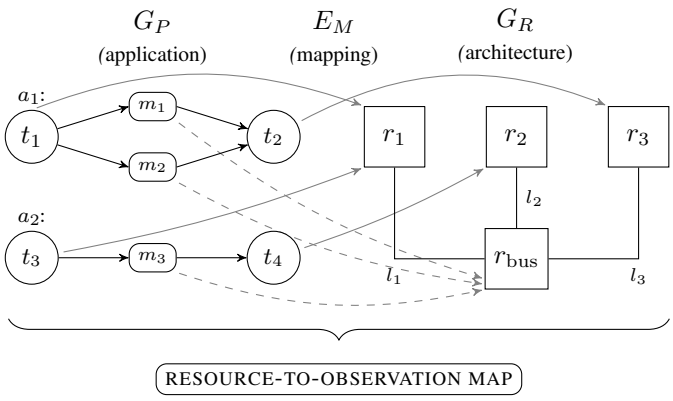

Fig. 5: Graph-based system specification. A process graph G_P consists of two applications a_1 and a_2 and is mapped onto a system architecture. The latter is defined by the resource graph G_R. The specifications are used to define resource-to-observation maps necessary for a correct distribution of detection tasks.

graph $G_P = (P, E_P)$ consists of the vertices $P = T \cup M$ corresponding to tasks $t \in T$ and messages $m \in M$, respectively, as well as the directed edges $e \in E_P$ representing their data-dependencies. The resource graph $G_R = (R, E_R)$, contains all system resources $r \in R$, such as ECUs and buses, as well as the appropriate links $l \in E_R$ between all architectural components. Both graphs are interconnected through mappings $E_M = (P, R)$ which associate each process with a particular resource.

We regard the system specification as given and our goal is to monitor the system in an implicit manner, which means that no additional diagnostic messages or hardware should be used. At this level of abstraction, the detection functions are implemented on multiple resources and can be regarded as tasks without any data-dependencies. In order to properly distribute the detection tasks with respect to both the redundancy requirements and a complete coverage of the monitored message streams, a preliminary system analysis is performed. The purpose is to define all observations describing the message streams and their parameters which can be monitored by a particular resource, not necessarily the transmitting one. These observations are then stored in a resource-to-observation map which basically associates sets of message stream data with the appropriate monitoring resources. Later, the resource-to-observation map is used to retrieve the necessary configuration parameters for the distribution of detection tasks, which is discussed in detail in Section IV.

III. Attack Detection

In hard real-time systems, an important precondition is the ability to react to an asynchronous event within a predefined time interval, i.e., without violating the deadline [18]. In order to detect an attack, such as an unusual or unwanted communication behavior, within a guaranteed or shortest possible detection time, each message stream in the system must be continuously monitored. Based on the observed occurrence of message stream events, resulting, for instance, in a violation of the upper arrival curve, particular security attack scenarios can be detected.

A. Considered Attacks and Limitations

There are fundamental differences between a classic computer network and an automotive in-vehicle network, for example, regarding the safety-criticality or the predictability in the system communication. Nevertheless, as shown in [19], the three general security principles, namely confidentiality, integrity and availability, can be applied in the automotive domain as well. A *confidentiality breach* enables the read access of messages by an unauthorized entity, and an attack may hack a resource allowing read access to the message content without producing additional traffic. An *integrity breach* enables an unauthorized creation or manipulation of messages. A possible attack may hack a resource to pretend as another resource and inject compromised messages into the network (e.g., with the message ID belonging to a stream from that resource). Finally, an *availability breach* interrupts one or more existing message streams. As illustrated in the motivating example, an attack could force one or multiple hacked resources to transmit a large number of (possibly void) messages in order to reduce the system's availability or overwhelm it. In this work, we consider availability attacks.

Our attack detection does not claim sufficiency or represent a universal security strategy for automotive E/E architectures. For example, attacks targeting confidentiality, such as message sniffing, and specific integrity breaches, such as single message injection, might not alter the traffic in a way that is detectable with the proposed algorithm. Consequently, many attacks will require own special security solutions which are not covered here. The presented approach targets a class of attacks, which result in an abnormally altered frequency of transmitted messages, where the attacker has no information about the impacted message arrival curves.

Due to its specific attack detection principle, this framework can improve automotive safety and reliability. For instance, it can be combined with the message-based fault diagnosis into a unified safety and security framework [20]. Nevertheless, because of the realistic threats arising from a malicious manipulation of traffic patterns (e.g., illustrated in Section I or in [4]) as well as its non-intrusive and lightweight detection principle the proposed approach is legitimate as an additional security solution orthogonal to other methods.

B. Real-Time Attack Detection

Figure 2, shows that the component-level part of the security framework detects compromised message streams. The detection method is explained by means of a common attack, namely the manipulation of the communication behavior. Figure 6 illustrates such an attack scenario, with the shaded area marking the compromised region and $R_{\widetilde{m}}(t)$ representing the message count from one possible stream violating the upper bound. This stream equates to the event trace function in Figure 3 depicted by the short red arrows. Based on the arrival curve definition in Equations (3a) and (3b), we can formulate a condition for the transgression of the upper bound, caused by an increased occurrence of message events. The attack requirement is defined in Equation (5).

$\exists t \in \mathbb{R}^+ :$

$$\alpha_m^u(\Delta) < \max_{t \geq 0} \{R_m(t + \Delta) - R_m(t)\} \tag{5}$$

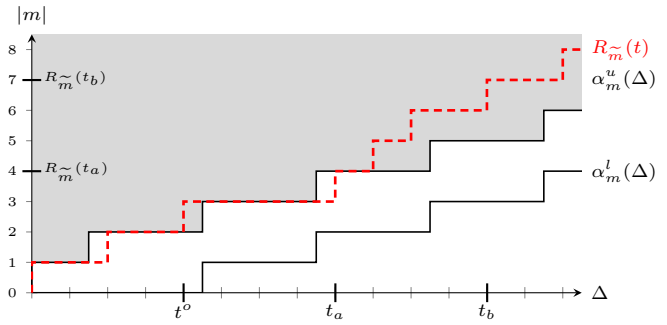

Fig. 6: Example for an attack on a single message stream by manipulating its transmission period. The corrupted message stream $\widetilde{m}$ (dashed red curve) violates the upper arrival curve α^u_m for the first time at t^o.

$R_m(t)$ is a message stream in normal runtime in contrast to an arrival curve, which resides in a time interval domain. For example, while $R_m(t)$ depicts messages with decreasing periods between them, this is not allowed in an arrival curve. Consequently, one cannot compare both curves directly for a limited number of message arrivals. To guarantee the detection of the earliest upper bound violation, it is necessary to continuously monitor the inter-arrival times of the message events. The algorithm for the detection of arrival curve violations is explained in detail in Section IV-A.

C. Redundancy-Aware Distribution of Attack Monitors

Due to the decentralized nature of our approach, different attacks may be detected by different monitors with varying, yet timely, detection times. It might not be necessary to consider all observable and potentially manipulated message streams on each monitor but only a subset of them, clearly reducing the implementation and computation effort. A design-time determination of monitors can help the designer to chose suitable monitors and configurations for particular message stream observations. In addition to the elimination of a single point of failure and the reduction of the computation effort, different redundancy levels can be used by using multiple monitors for each message stream.

For automatically distributing detection tasks, the system designer can configure two parameters: redundancy level λ_m for critical message streams as well as tolerance limits θ_r for the monitoring resources. Increasing λ_m improves the reliability of the attack detection and adjusting θ_r balances the distribution of monitors. Consider the architecture in Figure 7 with three ECUs, two buses and a gateway. Our detection method monitors the message streams m_1, m_2 and m_3 transmitted on bus_1 and bus_2. A compromised message stream can be detected by its associated detection tasks. For instance, task t_{m_1} can detect a manipulation of message stream m_1, task t_{m_2} a manipulation of message stream m_2, and so on. The left side of the figure shows a scenario with a redundancy level $\lambda_m = 2$. The tasks are not evenly distributed and there exist two detection tasks per message stream. By contrast, the optimized scenario on the right has a redundancy level of $\lambda_m = 3$. The system uses three detection tasks per message stream all of which are evenly distributed and, hence, use the

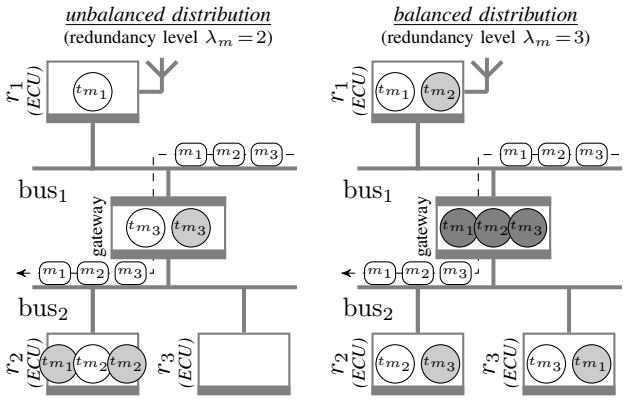

Fig. 7: Example for different distributions of detection tasks. The left-hand side architecture uses two tasks per message stream which are unevenly distributed yielding a redundancy of $\lambda_m = 2$. In the right-hand side architecture the tasks are evenly balanced with a higher redundancy level of $\lambda_m = 3$.

resources more equitably. The distribution process is formally described in Section IV-B.

IV. Methodology

We consider two important metrics for the attack detection method, namely *timeliness* and *coverage*. Timeliness guarantees a minimal detection time which is essential in safety-critical real-time systems. Coverage ensures that all detectable attacks are correctly and reliably detected. This section provides the formal descriptions of the real-time attack detection and the distribution of attack detection tasks.

In the following, $\widetilde{m}$ is used to denote the manipulated version of the message stream m and is accordingly described by the period $p_{\widetilde{m}}$ and a worst-case jitter $j_{\widetilde{m}}$.

A. Real-Time Attack Detection

Since a timely detection of an attack is crucial in a safety-critical system, an obvious approach is to test for arrival curve violations of message streams immediately after the arrivals of the corresponding messages. At the monitoring resource, an event trace function $\mu_m(n)$ provides timestamps t_{n-1} for message instances of the stream m (see Equation (1)).

The correct and timely detection requires the continuous monitoring of message inter-arrival times. Consequently, in the example in Figure 6 checking the violation for each message arrival separately could not detect an attack before the first arrival after time t_a. By contrast, considering at least two consecutive message arrivals would allow to detect the violation at time t^o. Clearly, a sufficient observation time is important when considering a bursty behavior of the message stream with the upper arrival curve steeply ascending in the beginning before swinging into a periodic run. At the same time, checking for an attack only after detecting a particular number of message violations is not a good strategy, since a message count can fall below α^u before exceeding it again, as demonstrated in Figure 6 around time t_a. It is important to consider the history of a monitored message stream, i.e., the arrival times of the previous messages.

Algorithm 1 Detection of compromised message streams by a resource r

Require: $\bigcup_{m,s}(\delta_{m,s}, \nu_{m,s})$ ▷ *tuples describing the arrival curves α^u_m for all message streams m monitored by resource r*

1: **for** $s \in \{1,..,n\}, m \in M_r$ **do** ▷ *initialize timers and counters*
2: $\text{TIMER}_{m,s} \leftarrow \delta_{m,s}$
3: $\text{COUNTER}_{m,s} \leftarrow \nu_{m,s}$
4: **end for**
5: **for** $m \in M_r$ **do** ▷ *analyze monitored message streams*
6: **if** $\text{received}(m) = \text{TRUE}$ **then** ▷ *message from m arrived*
7: **for** $s \in \{1,..,n\}$ **do**
8: **if** $\text{COUNTER}_{m,s} = \nu_{m,s}$ **then** ▷ *first message in $\delta_{m,s}$*
9: $\text{TIMER}_{m,s} \leftarrow \delta_{m,s}$
10: **end if**
11: $\text{COUNTER}_{m,s} \leftarrow \text{COUNTER}_{m,s} - 1$ ▷ *reduce message counter*
12: **if** $(\text{COUNTER}_{m,s} < 0)$ **then**
13: $\text{attacked}(m)$ ▷ *report attack on m*
14: **end if**
15: **end for**
16: **end if**
17: **for** $s \in \{1,..,n\}$ **do** ▷ *adapt message counters after timeout*
18: **if** $\text{timeout}(\text{TIMER}_{m,s}) = \text{TRUE}$ **then**
19: $\text{COUNTER}_{m,s} \leftarrow \min(\text{COUNTER}_{m,s} + 1, \nu_{m,s})$
20: $\text{TIMER}_{m,s} \leftarrow \delta_{m,s}$
21: **end if**
22: **end for**
23: **end for**

Leaky-bucket detection algorithm. Considering the issues discussed above, for the proposed attack detection we adopt the *leaky bucket* approach used to validate the runtime conformity of event arrivals in the context of network calculus [21]. The method is lightweight and can be efficiently implemented, which makes it ideal for hard real-time systems. The basic idea is to monitor the numbers of message arrivals in all time intervals that define an upper arrival curve.

Algorithm 1 illustrates the detection procedure by resource r monitoring message streams $m \in M_r$ with given upper arrival curves $\alpha^u_m(\Delta)$ using the following parameters.

p_m	nominal period of a message stream m
j_m	worst case jitter in p_m
$\delta_{m,s}$	minimum time interval between consecutive messages in a staircase function s
$\nu_{m,s}$	maximum number of messages within interval length zero (burst capacity)
$\rho_{m,s}$	message counter decrement

The algorithm is based on the event stream regulation from [22] and assumes that each arrival curve can be approximated by a minimum on a set of periodic staircase functions [23]. An arrival curve with distinct step widths can be represented by n tuples $(\delta_{m,s}, \nu_{m,s})$, $s \in \{1,..,n\}$, which are in ascending order with respect to $\nu_{m,s}$.

The initialization steps (lines 2–4) allocate time intervals $\delta_{m,s}$ and the corresponding initial message numbers at time zero $\nu_{m,s}$ to timers ($\text{TIMER}_{m,s}$) and message counters ($\text{COUNTER}_{m,s}$), respectively, for each considered staircase function s and message stream m. For this reason, $\text{COUNTER}_{m,s}$ represents the capacity. During the detection process (lines 5–23), the algorithm checks for arriving messages (line 6) and expired timers (line 18) of each message stream m. Upon arrival of a message, it is then tested if each counter is still full and, thus, if m was the first message within the time interval $\delta_{m,s}$ (line 8), in which case the corresponding timer will be restarted (line 9). The latter step guarantees that a potential violation of the upper arrival curve will be detected and can be visualized by aligning the beginning of the arrival curve to the first message of the count R_m as depicted, for instance, in Figure 6. Subsequently, the counters are decremented and an arrival curve violation on the message stream m is reported if any of them falls below zero (lines 11–14). After each expiration of $\text{TIMER}_{m,s}$, the $\text{COUNTER}_{m,s}$ values are adapted and the timers are reset to the appropriate time intervals (lines 17–22).

Correctness of Algorithm 1 can be derived from the formal proofs in [22].

Since the algorithm iterates over both message streams and distinct staircase functions, it has $\mathcal{O}(n^2)$ complexity. However, its efficiency can be improved by leveraging the periodicity of the message-based communication in automotive networks, where the boundaries of message streams can often be described by arrival curves composed of a nominal period p and a worst case jitter j. Based on the derivations for horizontally shifted staircase functions [23], we can adapt the algorithm for a periodic message stream with jitter, as has been defined in Equation (4). It needs one tuple as input including an additional parameter ρ describing the message counter decrement. The three parameters are defined in Equation (6), with $\gcd(x_1, x_2)$ returning the *greatest common divisor* of x_1 and x_2.

$$\begin{aligned} \delta_m &= \gcd(p_m, p_m - j_m) \\ \nu_m &= 2 \cdot \rho_m - \frac{p_m - j_m}{\delta_m} \\ \rho_m &= \frac{p_m}{\delta_m} \end{aligned} \tag{6}$$

After initializing the TIMER_m with δ_m and the COUNTER_m with ν_m, only line 11 has to be modified to decrement the counter by ρ_m rather than by 1 ($\text{COUNTER}_m \leftarrow \text{COUNTER}_m - \rho_m$). The modified algorithm requires only one timer variable per monitored message stream (i.e., $s \in \{1\}$) resulting in a reduced complexity. The algorithm scales linearly and can be efficiently implemented with a reduced memory and runtime overhead.

To illustrate the procedure, consider the communication example in Figure 1, where a message stream m with period $p_m = 30$ ms and jitter $j_m = 15$ ms is attacked and altered to yield a compromised message stream $\widetilde{m}$ with $p_{\widetilde{m}} = 20$ ms and no jitter. Applying the given period and jitter to Equation (6) leads to the parameters for the detection algorithm $\delta_m = 15$ ms, $\nu_m = 3$ and $\rho_m = 2$. This means that TIMER_m is loaded with 15 and COUNTER_m with 3 and decremented by 2 each time a message arrives. When neglecting the actual computation times and assuming that the first message of $\widetilde{m}$ is detected instantly after the algorithm starts, then the detection time (i.e., the time until the method $\text{attacked}(m)$ is reached)

is 40 ms. The fast execution ensures that an attack is detected early in order to initiate recovery counter measures.

B. Redundancy-Aware Distribution of Attack Monitors

The proposed security approach is designed for a decentralized implementation where the attack detection algorithm simultaneously monitors multiple message streams on different resources. On the one hand, this approach guarantees continuous detection in case of a system fault or attack that disables one or more resources. On the other hand, it allows the system designer to evenly distribute the computation amongst the resources, increasing the overall reliability and security. The decentralized detection covers the entire architecture, including remote parts of the system, such as different subnetworks. Moreover, in order to improve the security of safety-critical messages, redundant detection tasks are run on different resources to monitor considered message streams. An essential requirement is that each message transmitting resource, such as an automotive bus, should be connected to at least one computation resource (e.g., ECU, gateway), enabling monitoring of the message streams.

We use the graph-based specification of the system architecture. As described in Section II-C, the graph $G_R = (R, E_R)$ connects resources $r \in R$ (e.g., ECU and buses) through architectural links $l \in E_R$ where the latter are defined as resource pairs (r_i, r_j). Sets R_{bus} and R_{ecu} contain the bus and the computational resources, respectively. A binary function $\text{detect}(r)$ indicates if a resource r supports attack detection tasks ($\text{detect}(r) = 1$) or not ($\text{detect}(r) = 0$)[4]. The system-level coverage of attack detection is defined in Equation (7).

$$\begin{aligned} &\forall m \in M : \\ &\quad \exists r \in R_{ecu}, \exists r_b \in R_{\text{bus}} : \\ &\qquad \sum \text{detect}(r) \wedge (r, r_b) \in E_R \wedge (m, r_b) \in E_M > 0 \end{aligned} \tag{7}$$

According to the equation, for each message stream m there exists at least one resource that monitors it for attacks and contains an architectural link $l \in E_R$ to the bus resource r_b serving the message stream m. This is the minimal requirement for the system to detect all message stream manipulations.

The decentralized attack detection approach reinforces reliability as well. This is achieved by redundant monitoring of each message stream and exploiting the fact that bus-attached resources can monitor messages which are not destined for them. Our approach seeks to evenly distribute the computation effort across the monitoring resources while maintaining a predetermined redundancy level λ_m for each message stream. This is formulated as an Integer Linear Program (ILP). The ILP and its corresponding parameters are defined below.

$M_r \subseteq M$	subset of message streams which are observable by a resource $r \in R$
$\lambda_m \in \mathbb{N}$	*redundancy level*, defines the number of monitoring resources for a message stream m
$\overline{\lambda_r} \in \mathbb{R}$	average redundancy level on a resource $r \in R$
$\theta_r^l, \theta_r^u \in \mathbb{R}$	*tolerance limits*, define the lower and upper limit for the monitoring messages around the system average $\theta_r^l \leq \frac{\lvert M \rvert}{\sum_{\forall r \in R} \text{detect}(r)} \leq \theta_r^u$
$\mathbf{o}_{r,m} \in \{0, 1\}$	*observation switch*, determines if a message stream m observable by resource r will be monitored (1) or not (0) at runtime
$\mathbf{x}_r \in \mathbb{N}$	*monitoring variable*, determines the number of monitored message streams on resource $r \in R$
$\mathbf{y}_r \in \{0, 1\}$	*range switch*, determines if the monitoring variable $\mathbf{x}_r$ resides in the desired range

$$\underset{\mathbf{y}_r \in \{0,1\}}{\text{maximize}} \sum_{\forall r \in R} \mathbf{y}_r \tag{8a}$$

subject to:

$$\forall m \in M : \sum_{\substack{\forall r \in R \\ \wedge m \in M_r}} (\text{detect}(r) \wedge \mathbf{o}_{r,m}) = \lambda_m \tag{8b}$$

$$\forall r \in R : \sum_{\forall m \in M_r} (\text{detect}(r) \wedge \mathbf{o}_{r,m}) = \mathbf{x}_r \tag{8c}$$

$$\forall r \in R : \quad \mathbf{x}_r \leq |M_r| \tag{8d}$$

$$\forall r \in R : \quad \mathbf{x}_r \leq \theta_r^u \cdot \overline{\lambda_r} \cdot \mathbf{y}_r + |M_r| \cdot (1 - \mathbf{y}_r) \tag{8e}$$

$$\forall r \in R : \quad \mathbf{x}_r \geq \theta_r^l \cdot \overline{\lambda_r} \cdot \mathbf{y}_r \tag{8f}$$

Constraint (8b) uses the *redundancy level* λ_m to determine the number of resources that monitor the message stream m. For instance, for a redundancy level $\lambda_m = 1$ each message stream is monitored by one resource. Constraint (8c) uses a *monitoring variable* $\mathbf{x}_r$ to determine the number of message streams on each resource r which are simultaneously monitored. $\text{detect}(r)$ indicates the attack detection capability of the resource r. As shown in Constraint (8d), the monitoring variable is bounded from above by the cardinality of M_r, namely the set of all streams observable by r. Constraints (8e) and (8f) define the range for the monitoring variable and, thus, the number of detection tasks on each resource. The *tolerance limits* $\theta_r^{u/l}$ determine how much the number of monitors can deviate from the system-wide average, where the latter is calculated as the ratio of all monitored message streams to all monitoring resources. Since Constraints (8e) and (8f) must consider redundancy levels greater than one, the tolerance limits are multiplied by the average redundancy level $\overline{\lambda_r}$, the arithmetic mean of the redundancy levels of all message streams monitored by one resource. The tolerance limits $\theta_r^{u/l}$ and the values of $\overline{\lambda_r}$ are precomputed. The *range switch* $\mathbf{y}_r$ decides if the number of monitored streams is within the predefined range ($\mathbf{y}_r = 1$) or not ($\mathbf{y}_r = 0$). The objective is to maximize the sum of range switches over all resources and it should yield a balanced monitoring of message streams (Equation (8a)).

V. Experimental Results

In this paper, we are interested in investigating the feasibility and efficiency of our attack detection method. As a convincing

[4]Reasons for a resource being incapable of detection can range from insufficient computational performance to the lack of fundamental architectural capabilities, e.g., in case of a bus resource.

proof of this concept, after a brief description of the test environment, we use a test case evaluation to demonstrate both the timely detection of manipulated message streams and the automated distribution of detection tasks. A runtime analysis shows the efficiency and scalability of our framework. Finally, a case study illustrates our approach in more detail on a large E/E architecture with more than 100 ECUs.

A. Test Environment

460 synthetic test cases were generated, analyzed and evaluated. Each test case contains the graph-based description for one specification including the E/E system architecture, the applications and the mappings between them. In an architecture graph the vertices represent E/E hardware, such as ECUs and gateways, while the edges represent the communication-links. In an application graph, vertices represent tasks and messages and edges represent dependencies between them. Although the test cases are synthetic, their topologies, the numbers and distribution of tasks and messages as well as the timing parameters reflect real automotive E/E architectures. Some of their parameters together with their minimum and maximum values are listed in Table I. The test cases encompass E/E architectures of high-end vehicles with over 100 ECUs.

All experiments were carried out on an Intel Core i7 with 3.4 GHz and 16 GB RAM. The system specifications were generated with the help of OPENDSE [24], a Java-based design space exploration framework for embedded systems, and all ILP-based optimizations used GUROBI version 6.0 as solver [25]. The parameters above describe the hardware of our simulation environment used to evaluate and validate our work. They do not represent real automotive hardware which is usually significantly slower. A discussion on the expectable computational and memory overhead of real automotive hardware can be found in Section V-D.

B. Evaluation

All test cases are evaluated with respect to the time to detection, the efficiency and coverage of the detection task distribution as well as the corresponding computation times.

Timeliness. Figure 8 illustrates the message periods and time to detection averaged over all message streams in each test case. The graph depicts the predefined nominal periods (cross marks), the attack periods (i.e., increased message rates) (ring marks) and the times to detection (solid curve), respectively. For each message stream, the attack period has been randomly generated in the range $5 - 90\%$ of the nominal period. Due to the normal distribution of the random variable, the associated average values of the attack periods are located roughly at 50% of the nominal period curve and show a similar trend in the graph. The test cases are sorted by increasing average times to detection. The graph indicates a dependency between the time to detection and both the nominal message stream periods and the attack periods. On average, the times to detection are in the range of twice the attack periods which is in accordance with the detection algorithm. In this case, due to the normal distribution of the randomly generated attack periods, the blue line lies roughly in the range of the nominal periods and above their average for higher test case ids.

TABLE I: Reference values of parameters for 460 test cases.

	ECUs	*Buses*	*GWs*	*Tasks*	*Message streams*	*Nominal period [ms]*	*Attack period [period ratio]*
MIN	2	1	0	6	3	5	5%
MAX	117	4	1	366	438	100	90%

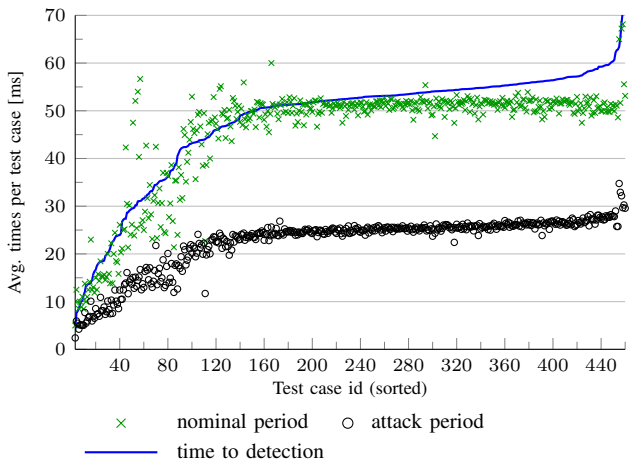

Fig. 8: The graph shows the average nominal periods (×) and attack periods (○) of the message streams. The solid curve illustrates the average times to detection of a potential attack. The results for the test cases are ordered by increasing average times to detection.

The overall outcome in Figure 8 indicates that a real-time detection of manipulated message streams using the introduced method is feasible. The illustrated times to detection represent the time intervals necessary to detect an attack. Since these results are simulated and do not consider potential computational overhead, it is important to test the efficiency of the security approach on real automotive hardware in future.

Coverage. Figure 9 shows three graphs depicting the average amount of monitored message streams per resource for each test case (thick blue curve). Coverage is calculated according to the ILP defined in Section IV-B. The vertical bars around this curve represent the appropriate standard deviation from the average values and, hence, are an indicator for how evenly the monitored messages are distributed among the monitoring resources. Lower standard deviations, i.e. shorter vertical bars, correspond to a more even distribution and vice versa. The aim of this graph is to visualize the variation of detection task numbers in relation to all other test cases as well as the three tolerance levels. The specific amounts of detection tasks are depicted more clearly for the case study in Section V-C. A second plot (thin green curve) in each graph shows the percentage of the results with a correct adherence to the predefined monitoring range (i.e., cases where $\mathbf{y}_r = 1$). To evaluate the effect of the tolerance limits θ_r, the curves are depicted for three desired ranges around the average message per resource number, namely $\pm 5\%$ (top), $\pm 20\%$ (center), and $\pm 50\%$ (bottom).

When comparing the compliance curve with the message stream number curve, a lower percentage of hits for the monitoring range is met by a higher standard deviation. This correct

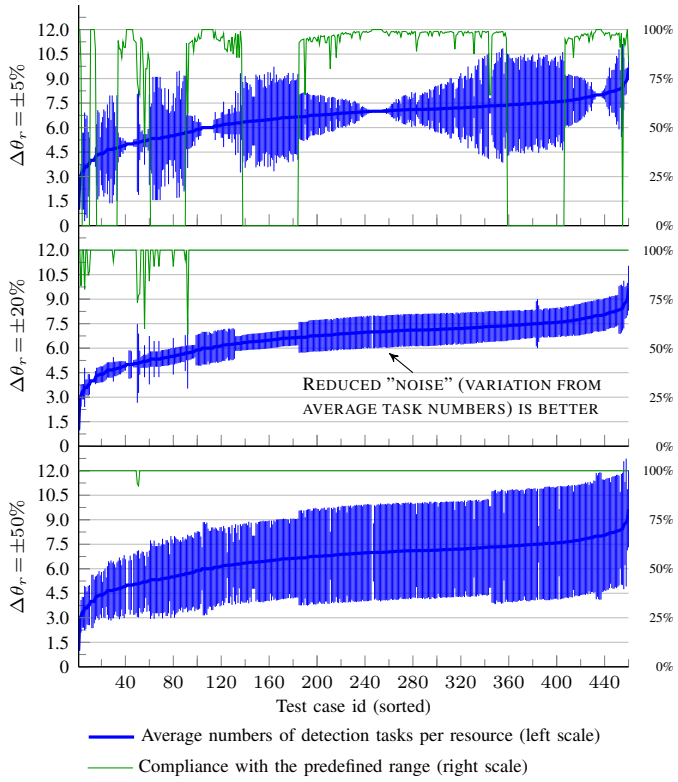

Fig. 9: The three graphs show the average number of monitored message streams per resource together with their standard deviation for each test case (thin vertical lines). A second curve indicates the compliance with the predefined monitoring range (right-hand side y-axis). The curves are plotted for three different tolerance values. The results are shown for a redundancy level $\lambda_m = 2$.

and presumed correlation verifies the ILP formulation and allows us to estimate an appropriate tolerance level. Obviously, while a small θ_r in many cases prevents the solver from finding an optimal solution (compare upper graph), a large value results in an overall high standard deviation (compare lower graph). From the three selected ranges, the middle one ($\pm 20\%$) seems to lead to the smallest standard deviation among all test cases and, hence, to the most balanced distribution of the monitored message streams among all resources. Although for particular architectures the lowest tested tolerance level can indeed lead to an optimal detection task distribution (e.g., test cases around 110 and 250), a level of $\pm 20\%$ provides satisfying results for a broad range of system specifications and could be used by the system designer as a starting point for a fine-grained adjustment. Considering the number of redundantly monitored message streams, all results in Figure 9 are calculated for a redundancy level $\lambda_m = 2$. A lower ($\lambda_m = 1$) and a higher ($\lambda_m = 3$) level have been investigated as well, and in both cases the overall tendency discussed above remains similar.

Runtime. Regarding the timing behavior, a short time to detection is important for the security approach and has been discussed by means of Figure 8. Within our security framework, both the algorithm-based attack detection simulation and the ILP-based distribution of detection tasks are performed at design-time, and, hence, do not impact the runtime.

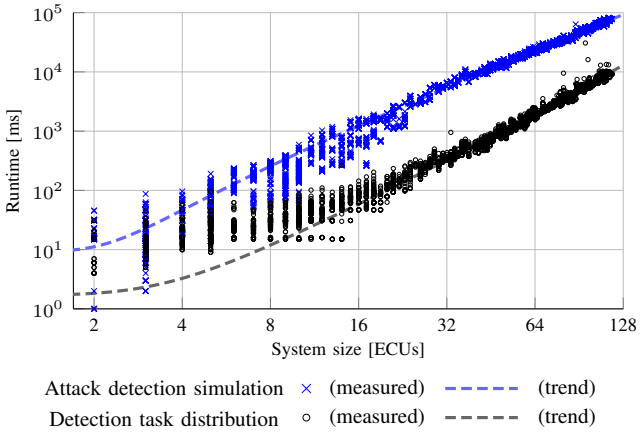

Fig. 10: Performance of the framework for different system sizes. The graph illustrates the computation times for the ILP-based distribution of attack detection tasks (including different test case configurations) (○) and the attack detection simulation (×). Additionally, two trend curves indicate a polynomial computational complexity.

Nevertheless, like most processes in the automotive industry, system-design is cost-sensitive and, thus, low computation times are beneficial. To investigate this, a runtime analysis has been performed and its results are shown in Figure 10. Here, the computation times of the simulation and distribution are regarded separately and plotted as a function of the system size depicted by the number of ECUs in the E/E architecture. Each measured point represents the runtime for one test case with a particular parameter configuration where the largest analyzed architecture is composed of 117 ECUs. In order to better visualize the distribution and behavior of the different runtimes, two dashed curves illustrate trends for both the simulation and the distribution, respectively[5]. The calculated trends are polynomial curves indicating a quadratic and cubic complexity in the number of ECUs for the simulation and distribution, respectively, when moving towards advanced architectures. At the same time, in general, the duration of the computation does not exceed 100 seconds even for the largest analyzed architectures. The longest simulation takes 83.1 seconds and the longest distribution 30.7 seconds. Considering that today's modern automotive networks consist of a lower three-digit number of ECUs, the results illustrate practical computation times of our security framework and indicate a good scalability for larger architectures in the future.

Note, that both axes in Figure 10 are in logarithmic scale and the numbers of test cases for different system sizes may vary, causing noticeable changes in the distribution of the single measurement points.

C. Case Study

An automotive case study shall illustrate the functionality of the proposed distributed attack detection in detail. The study is based on a state-of-the art automotive E/E architecture which consists of 144 ECUs[6], five buses and one automotive gateway. The system uses 618 tasks which communicate via 321 messages with message periods ranging from 5 to 100 ms and

[5]The approximation has been determined by the method of least squares.

message jitters determined according to the delay calculation for event-triggered systems proposed in [26].

The results are shown in Figure 11 where for each of the four graphs, monitoring resources are depicted on the x-axis and the number of monitored message streams per resource on the y-axis. The bars in each graph stand for a different tolerance level, namely 5% (top graph) 10% (second graph), 20% (third graph), and 50% (bottom graph). The two shades represent redundancy levels, $\lambda_m = 3$ (plain bars), and $\lambda_m = 1$ (striped bars), for which each message stream is monitored by three resources and one resource, respectively. An increased height of the plain bars indicates a higher number of detection tasks for the higher redundancy level. Here, the semi-transparent black curves lain on top of the bars shall help visualize the variations in the number of detection tasks for different resources.

For the two middle graphs with tolerance levels of 10% and 20%, the detection tasks for the different redundancy levels are evenly distributed among the monitoring resources, with approximately 7 and 3 tasks per resource for $\lambda_m = 3$ and $\lambda_m = 1$, respectively. However, for $\lambda_m = 1$, the second graph shows a slightly larger variance leaving a tolerance level of 20% as the most optimal option. By contrast, the graphs illustrating a tolerance level of 5% and 50% show much larger irregularities with up to 14 tasks difference between the resources. This study illustrates how a proper adjustment of the optimization parameters during the system design can help to evenly allocate the detection tasks to monitoring resources.

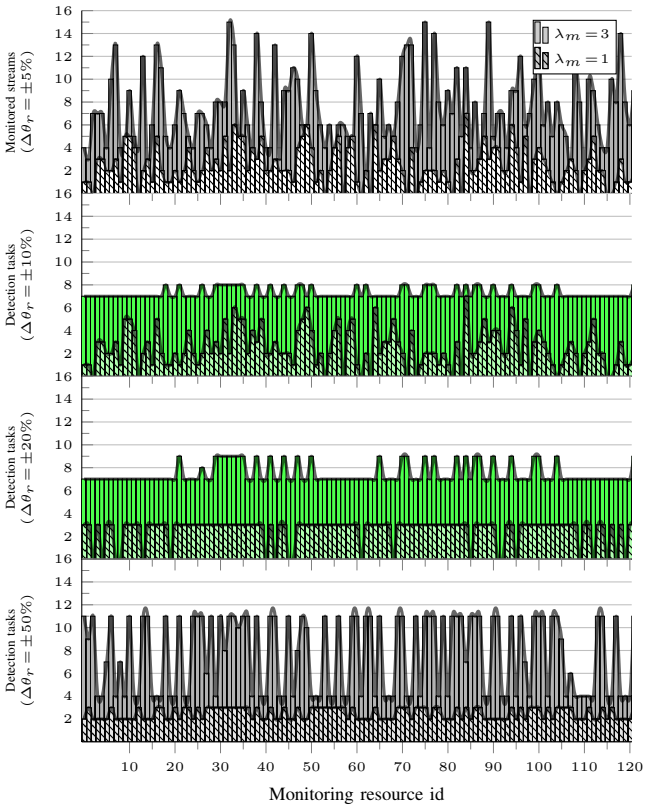

Fig. 11: The four graphs show the number of monitored message streams allocated to different resources for an exemplary automotive E/E architecture with 144 ECUs. The distribution is most balanced for the tolerance levels $\pm 10\%$ and $\pm 20\%$, as depicted in the two middle graphs. It varies largely for $\pm 5\%$ and $\pm 50\%$.

D. Overhead on Automotive ECUs

The results are based on the analysis and simulation of synthetic test cases as well as a synthetic case study which, as mentioned before, are based on current and future automotive (E/E) architectures with realistic system topologies and application parameters. The experiments were performed on laboratory hardware which provides a better performance than today's automotive ECUs. This is not an issue for the distribution of detection tasks since it would be done at design-time on fast computers. However, regarding the performance of the attack detection on real automotive hardware, we are not able to provide real measurements results at this time.

In the following we want to give an estimation of the computational overhead caused by the implementation of the attack detection on real automotive hardware. For this, we orient ourselves on the implementation results from a previous work addressing the distributed diagnosis of permanent faults in automotive networks [20]. Although the fault diagnosis algorithm is different in its objective and approach, it is comparable regarding its length and complexity as well as the way it deals with multiple message streams. Relating to the fault diagnosis results, the proposed attack detection approach, monitoring 15 message streams per resource, would utilize about 5% of the processor resources and less than 10 KB of memory. Keeping the overhead low will facilitate the implementation of the attack detection. Moreover, the predominantly periodic communication in automotive networks enables us to efficiently store the arrival curves data. As discussed in Section IV-A, using the periodic with jitter model the detection requires only three parameters per arrival curve.

[6]As of the 144 resources 24 have no monitoring capability. For clarity only the results for the remaining 120 resources are depicted in Figure 11.

The evaluation of our approach presented here indicates a good compliance with the NHTSA guidelines [7]. On the one hand, the timely attack detection accords with the rapid detection and remediation capabilities that the automotive industry should establish in order to mitigate the safety risk to vehicle occupants and surrounding road users. On the other hand, the coverage of the entire E/E architecture and the application of redundant message stream monitoring would ensure that vehicle systems can take appropriate and safe actions, even when an attack is successful.

VI. Related Work

This section contains a comprehensive overview and discussion of the existing work in the area of automotive security, where the focus is on the detection of attacks on in-vehicle networks.

The automotive industry is more than ever driven by the technological progress, notably in terms of connectivity and communication. The security risks arising from this development are evident, and beyond the stage of mere thought

experiments or lab setups, as pointed out in [27]. An introductory insight and discussion of automotive security is presented in [3]. On the one hand, it illustrates the threats and the challenges arising from the growing complexity of automotive electronics and its interconnections to the outside world. On the other hand, it proposes potential concepts for future solutions comprising Ethernet-based networks and formal verification approaches. The related threats can range from a relatively harmless unauthorized access to the vehicle by cracking the key-less entry system [28] to the much more severe intrusion into an in-vehicle network and infiltration of its ECUs. This can interrupt safety-critical systems and endanger the lives of the traffic participants [1, 11, 2]. Using the example of an Electronic Throttle Control (ETC), [5] highlights the relevance and danger arising from a DoS attack. After having hacked the engine management ECU, the attacker can use malicious code to flood the CAN bus with a large number of spoofed messages and provoke a DoS on the ETC unit, thereby causing an uncontrolled acceleration. Multiple approaches examine automotive security including inter-vehicular networks, Vehicle-to-Vehicle and Vehicle-to-Infrastructure communication [30, 31]. [32] analyzes the performance of vehicular communication systems, where the car is a node in a network. [33] investigates IPv6-based protocol to secure a mobile vehicular communication and demonstrate that this is possible without serious network overload.

A general cybersecurity guidance has been recently proposed by the NHTSA [7]. It focuses on solutions to harden the automotive E/E architecture against potential attacks and ensures that safe actions are taken in case of successful attacks. A systematic and ongoing process to evaluate risks should be followed through the entire life-cycle of the vehicle including design, manufacture, maintenance and decommissioning. The proposed best practices advice the automotive industry to use a layered approach to vehicle cybersecurity which should be built upon risk-based prioritized identification and protection of safety-critical systems. Notably, the guidance advocates a timely attack detection followed by a rapid response to potential vehicle cybersecurity incidents and recovery from these incidents when they occur. The distributed traffic monitoring proposed in this paper complies with these practices.

Overall, cyber-attacks on automotive systems are a practical threat that must be considered both during the design of E/E architectures and during the operation time of the vehicle.

Several papers address automotive security at design-time. [34] integrates security into the traditional Design Space Exploration (DSE) of distributed embedded systems. In [35] the security in the DSE process is addressed but focuses on the mapping of functional models to CAN-based platforms which are additionally applying Message Authentication Codes (MACs). To improve the efficiency and define the realistic security scenarios, the work in [36] extends [35] by redefining the Mixed Integer Linear Programming (MILP) formulations to consider functional paths. The application of MACs for the Flex-Ray bus is presented in [37]. Verification of security requirements of automotive architectures is proposed in [19]. The approach uses security-specific parameters such as exploitability and patching rate to describe the system and determine a Markov model which is then subjected to a probabilistic model checking. Our security approach covers two tasks at design-time. First, a system analysis extracts the necessary parameters for the runtime detection and second, an optimal distribution of the detection tasks among the system resources is performed. Our work follows a rather different approach from the papers above, mainly because it uses the unaltered system specification for the attack detection process (i.e., without initial security extensions).

Together with its consideration during automotive system-design, security has been also addressed in the context of runtime attack detection. A general detection scheme for attacks on in-vehicle networks is proposed in [38], using a set of eight criteria for the identification of a typical behavior of automotive bus systems, such as CAN. The work presents an Intrusion Detection System (IDS) for future vehicles called *Anomaly Detection Sensors*, that inspects the communication in terms of formality, location, range, frequency, correlation, protocol, plausibility and consistency. [4] and [38] have proposed anomaly detection using an information-theoretic method. There, using an entropy-based increased frequency detection at runtime is related to our approach. However, the evaluation merely proves the concept and does not provide the observation times needed for the detection. Thus, a direct comparison with our method in terms of real-time efficiency is not possible. Beyond the automotive domain, the importance of a distributed IDS, which removes the single point of failure, has been discussed in [39]. The differentiation between normal and manipulated traffic is based on a feature extraction performed during a parametrization and training phase on each node using the *Naïve Bayes Classifier* and the agreement between the nodes is determined using an average consensus protocol. In contrast to this probabilistic method, our approach takes advantage of the fact that automotive system and communication properties are known at design-time, enabling a verification if the observed traffic behavior conforms to the given specification.

VII. Concluding Remarks

The overall goal of this work lies in improving the security of safety-critical distributed systems, such as automotive E/E architectures, in a decentralized and implicit manner by enhancing the reliability of the attack detection and reducing the implementation costs at design time. Motivated by a realistic attack scenario on an electric vehicle, we demonstrated the importance of timely, distributed and reliable detection which can be provided by our framework. The decentralized nature of the approach allows the detection algorithm to be efficiently implemented in existing E/E system architectures without the risk of a single point of failure, hence, increasing the overall reliability. We demonstrated the feasibility and efficiency of our approach on a number of test-cases including 460 synthetically generated system specifications with sizes up to 117 ECUs. Additionally, a case study with 144 ECUs provided a detailed investigation of the security framework including a realistic estimation of the computational overhead which lies below 5%.

As part of future work, we want to extend the optimization-based detection task distribution by allowing the designer to specify constraints, such as different criticality levels or

specific vulnerable system resources. Furthermore, we want to consider other security threats which might result in an alteration of the traffic patterns, such as the removal or overwriting of lower priority messages or the manipulation of the message content causing a potential delay of the sending instance.

REFERENCES

[1] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks–practical examples and selected short-term countermeasures," in *Computer Safety, Reliability, and Security*, pp. 235–248, Springer, 2008.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the USENIX Security Symposium*, 2011.

[3] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty, "Security challenges in automotive hardware/software architecture design," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 458–463, 2013.

[4] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *Proceedings of the Intelligent Vehicles Symposium (IV)*, pp. 1110–1115, 2011.

[5] T. Hoppe, S. Kiltz, A. Lang, and J. Dittmann, "Exemplary automotive attack scenarios: Trojan horses for electronic throttle control system (ETC) and replay attacks on the power window system," in *In Automotive Security - VDI-Berichte Nr. 2016, Proceedings of the VDI/VW Gemeinschaftstagung Automotive Security*, pp. 165–183, 2007.

[6] T. Hoppe and J. Dittman, "Sniffing/replay attacks on CAN buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy," in *Proceedings of the 2nd workshop on embedded systems security (WESS)*, pp. 1–6, 2007.

[7] N. H. T. S. Administration, "Cybersecurity best practices for modern vehicles." [Online]: available at http://www.nhtsa.gov/About-NHTSA/Press-Releases, 2016.

[8] S. Chakraborty, M. A. A. Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Design & Test*, vol. 33, no. 4, pp. 92–108, 2016.

[9] Nissan Motor Company Ltd., "Nissan LEAF owner's manual - section LAN systems." [Online]: available at https://carmanuals2.com/brand/nissan/leaf-2014-374, 2013.

[10] R. Roderick, "A look inside battery-management systems." [Online]: available at http://electronicdesign.com/power/look-inside-battery-management-systems, 2015.

[11] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, *et al.*, "Experimental security analysis of a modern automobile," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 447–462, 2010.

[12] U. Klehmet, T. Herpel, K.-S. Hielscher, and R. German, "Delay bounds for CAN communication in automotive applications," in *Proceedings of the Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pp. 1–15, 2008.

[13] S. Chakraborty, S. Kunzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 190–195, 2003.

[14] S. Kuenzli and L. Thiele, "Generating event traces based on arrival curves," in *Proceedings of the Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pp. 1–18, 2006.

[15] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," in *Proceedings of the International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pp. 62–69, 1999.

[16] T. Nolte, H. Hansson, and C. Norstrom, "Minimizing CAN response-time jitter by message manipulation," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 197–206, 2002.

[17] U. Suppiger, S. Perathoner, K. Lampka, and L. Thiele, "A simple approximation method for reducing the complexity of modular performance analysis," Tech. Rep. 329, ETH Zurich, 2010.

[18] A. Albert, "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems," in *Proceedings of the Embedded World Conference*, pp. 235–252, 2004.

[19] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, "Security analysis of automotive architectures using probabilistic model checking," in *Proceedings of the Design Automation Conference (DAC)*, pp. 38:1–38:6, 2015.

[20] P. Waszecki, M. Lukasiewycz, and S. Chakraborty, "Decentralized diagnosis of permanent faults in automotive E/E architectures," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2015)*, pp. 189–196, 2015.

[21] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*, vol. 2050. Springer Science & Business Media, 2001.

[22] K. Huang, G. Chen, C. Buckl, and A. Knoll, "Conforming the runtime inputs for hard real-time embedded systems," in *Proceedings of the Design Automation Conference (DAC)*, pp. 430–436, 2012.

[23] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems," *Design Automation for Embedded Systems*, vol. 14, no. 3, pp. 193–227, 2010.

[24] M. Lukasiewycz and F. Reimann, "OpenDSE - open design space exploration framework." [Online]: available at http://opendse.sf.net/.

[25] Gurobi Optimization, Inc., "Gurobi Optimizer Reference Manual," 2015. [Online]: available at http://www.gurobi.com/.

[26] M. Lukasiewycz, S. Steinhorst, and S. Chakraborty, "Priority assignment for event-triggered systems using mathematical programming," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 982–987, 2013.

[27] L. Ben Othmane, R. Fernando, R. Ranchal, B. Bhargava, and E. Bodden, "Likelihood of threats to connected vehicles," *International Journal of Next-Generation Computing (IJNGC)*, vol. 5, no. 3, 2014.

[28] S. Bono, M. Green, A. Stubblefield, A. Juels, A. D. Rubin, and M. Szydlo, "Security analysis of a cryptographically-enabled RFID device," in *Proceedings of the USENIX Security Symposium*, vol. 5, pp. 1–16, 2005.

[29] T. Hoppe, S. Kiltz, and J. Dittmann, "Adaptive dynamic reaction to automotive IT security incidents using multimedia car environment," in *Proceedings of the International Conference on Information Assurance and Security (IAS)*, pp. 295–298, 2008.

[30] X. Huang, J. Wu, W. Li, Z. Zhang, F. Zhu, and M. Wu, "Historical spectrum sensing data mining for cognitive radio enabled vehicular ad-hoc networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 1, pp. 59–70, 2015.

[31] C. Lyu, D. Gu, Y. Zeng, and P. Mohapatra, "PBA: Prediction-based authentication for vehicle-to-vehicle communications," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 1, pp. 71–83, 2015.

[32] G. Calandriello, P. Papadimitratos, J.-P. Hubaux, and A. Lioy, "On the performance of secure vehicular communication systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 898–912, 2011.

[33] P. Fernandez, J. Santa, F. Bernal, and A. Gomez-Skarmeta, "Securing vehicular IPv6 communications," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 1, pp. 46–58, 2015.

[34] I. Stierand, S. Malipatlolla, S. Froschle, A. Stuhring, and S. Henkler, "Integrating the security aspect into design space exploration of embedded systems," in *Proceedings of the International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 371–376, 2014.

[35] C. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli, "Security-aware mapping for CAN-based real-time distributed automotive systems," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 115–121, 2013.

[36] C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems," *Embedded Systems Letters, IEEE*, vol. 7, no. 1, pp. 11–14, 2015.

[37] G. Han, H. Zeng, Y. Li, and W. Dou, "SAFE: Security-Aware FlexRay scheduling engine," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 1–4, 2014.

[38] M. Müter, A. Groll, and F. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *Proceedings of the International Conference on Information Assurance and Security (IAS)*, pp. 92–98, 2010.

[39] M. Toulouse, B. Q. Minh, and P. Curtis, "A consensus based network intrusion detection system," in *Proceedings of the IT International Conference on Convergence and Security (ICITCS)*, pp. 1–6, 2015.