EasyChair Preprint

№ 3769

# Multilevel Runtime Security and Safety Monitoring for Cyber Physical Systems using Model-based Engineering

Smitha Gautham, Athira Varma Jayakumar and Carl Elks



July 6, 2020

# Multilevel Runtime Security and Safety Monitoring for Cyber Physical Systems using Model-Based Engineering

Smitha Gautham✉ '[0000-0001-5394-9832]', Athira V. Jayakumar, and Carl Elks

Virginia Commonwealth University, Richmond, VA, USA
gauthamsm@vcu.edu

**Abstract.** Cyber-Physical Systems (CPS) are heterogeneous in nature and are composed of numerous components and embedded subsystems that are interacting with each other and with the physical world. The interaction of hardware and software components at each level, expose them to attack surfaces, which need novel methods to secure against. To ensure safety and security of high integrity CPSs, we present a multilevel runtime monitor approach where there are monitors at each level of processing and integration. In the proposed multi-level monitoring framework, some monitoring properties are formally defined using Event Calculus. We then demonstrate the need for multilevel monitors for faster detection and isolation of attacks by performing data attack and fault injection on a Simulink CPS model.

**Keywords:** runtime monitors, Event Calculus, model-based engineering, cyber-physical systems.

## 1 Introduction and Motivation

Cyber Physical Systems (CPSs) are heterogeneous architectures composed of physical, network and computational components that are tightly integrated together that allow human cyber interactions [1]. To do this, CPSs are evolving toward *software intensive systems* where functionality, integration, and operations of a given system are largely governed by its complex software interactions. Although software testing methods and practices have undergone tremendous progress over the past 20 years, the evolving nature of software intensive CPSs can create layers of unforeseen failure modes and complex attack surfaces. These can lead to safety design assurance issues at design time and become problematic for ensuring safety at runtime. Such challenges (among others) are emerging drivers for new design and development and operation practices that strive to reduce cost without compromising safety – termed as *DevOps Safety Continuum* [2].

In many safety critical application domains, runtime monitors (or runtime verification) are used to enforce operational safety and security – as a complementary defense to design assurance [3]. Runtime monitors can be thought of as means to detect and mitigate failures/attacks that design time verification may have omitted or overlooked. In order to detect attacks and complex evolving failures across a CPS, we posit that single monitor solutions are insufficient. Rather we suggest that multiple distinct types of monitors positioned across the CPS provide more comprehensive detection and location capability [4]. This paper supports an important aspect of

DevOps Continuum; namely that such multilevel monitors are a promising consideration toward ensuring operational safety in complex CPSs.

We implemented the multilevel monitoring scheme in a MathWorks Simulink Model Based Engineering tool to ascertain the benefits and challenges of evaluating multilevel monitors with respect to security and safety considerations. Model-based Design and Engineering (MBDE) approaches are widely becoming the normative methodology to design, verify and validate safety-critical Cyber Physical Systems (CPS) across various domains (e.g. automotive and aerospace). Our practical and technical contributions that help in the design of dependable CPS are:

- Development of a novel multilevel monitoring framework for runtime safety and security monitoring of CPSs.
- Evaluating the efficacy of multilevel monitoring framework in detecting faults and attacks in a distributed CPS
- Use of MBDE tools to connect design time with runtime monitoring for accessing security and safety considerations early in the design development process.

## 2 Related Work

With the growth in use of CPSs in numerous safety critical applications, runtime verification of such systems is becoming an essential and important topic of research. Ref [5] presents a bus monitoring approach for COTS processors where the communication between the peripherals and the system are monitored. Ref [3] presents a bolt-on monitor that silently receive messages over a CAN bus without affecting the system functionality. With the limited information available on the bus, the runtime monitor verifies safe system behavior. Ref [6] present a non-intrusive monitoring approach for multi-core processors based on the execution traces received by the processors. A three layer CPS architecture is proposed in [7] comprising of transport layer, control layer and execution layer and attacks that can occur at each of these layers is surveyed but no specific example is provided. Ref [8] provides a comprehensive survey of monitoring distributed real time systems and provides architectural frameworks to monitor processor and Bus in a CPS. Monitors have been modeled using MBDE tools in prior work, for example Ref [9] uses Simulink to model runtime monitors. However, multilevel monitors for a CPS with specific attacks/faults has not been explored. Our paper contributes to this area by clearly demonstrating the need for multilevel monitors (data, network and functional) in a CPS to detect a wide range of attacks as well as locate their origin. We then evaluate our multilevel monitoring framework using MathWorks Simulink tools.

## 3 Development of a Multilevel Monitoring Framework

CPS are heterogeneous in nature encompassing many computational units and include physical interfaces to sensors and actuators. It is important to not just monitor each component in a CPS individually but also monitor the interaction of the components

and the physical environment. **Fig. 1** depicts a common interpretation of a generalized CPS structure [9].

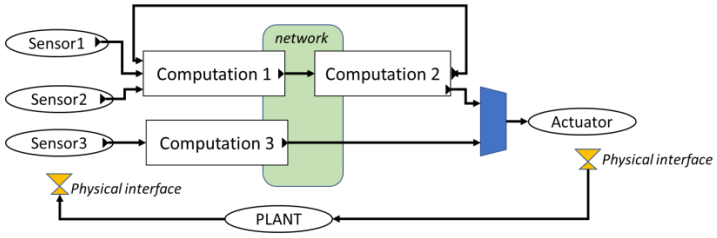


**Fig. 1.** Structure of a Cyber Physical System (Adapted from [10] but modified and redrawn).

Referring to **Fig. 1**, the attacks on a CPS can be broadly classified into three domains. First, attacks on low level *hardware/firmware-oriented* devices. These include sensor or actuator attacks, for example sensor spoofing, firmware attacks, replay attacks to name a few. Second, attacks on the *connection* or *network* layer (e.g. I2C, CAN, SPI) that include attack on a communication bus such as Denial of Service (DoS), packet injection, eavesdropping. Lastly, attacks on the *computational elements* such as malware injection, control flow attack, buffer overflow etc. that can affect the functionality of the processing unit. In this paper we consider attacks and faults that affect *hardware*, *network* and *computational* elements in a CPS and architect a multilevel runtime monitoring framework to effectively detect and isolate the origin of the attack/fault. We consider three levels of monitoring across a CPS. They are:

- Data monitors: They mainly monitor the *hardware/firmware-oriented* devices such as sensors and actuators that constantly interact with the outside environment. They check for integrity of the information coming from these devices through the physical interface.
- Network monitors: They mainly monitor the *connection or network* layer of the CPS. Sensors, actuators and computational units in a CPS use communication protocols such as UART, I2C and buses such as CAN. Network monitor checks for signal faults, incorrect signaling protocol, timing, configurations, etc. in these communication networks.
- Functional monitors: They mainly monitor the *computational* units of a CPS to verify the overall system behavior or functionality of a processing unit within the CPS. Safety and security properties are monitored for expected system behavior.

Having monitors at multiple levels (data, network and functional monitors) should ensure that more classes of faults/attacks can be detected and isolated early before it propagates and affects the system (**Fig. 2**). Attacks that fall outside the intersection, in **Fig. 2** can only be detected by having a localized monitor at that particular level in the CPS. Having these local monitors at each critical level in a CPS helps cover one other's blind spot [11]. We demonstrate the benefits of multilevel monitoring scheme with the specific example of an Anti-lock Braking System (ABS). Furthermore, we show that some faults/attacks may be detected by monitors at other levels (than that of their

origin). Even in such cases, monitors at multiple levels are needed to find the location of these faults/attacks.

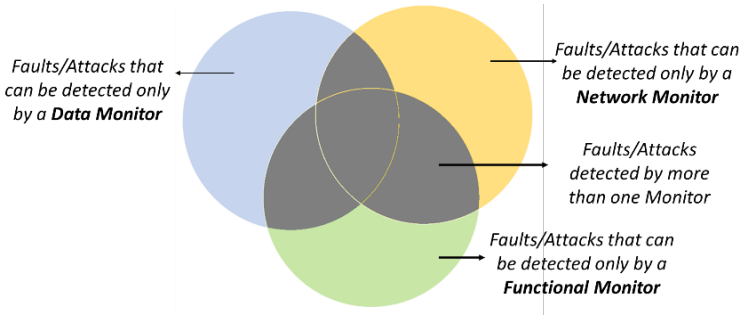


**Fig. 2.** Attacks/Faults detected by multilevel monitors.

## 4 Example CPS: Anti-lock Braking System (ABS) and Event Calculus to Specify Safety and Security Properties

We use a Simulink model of an Anti-lock Braking System (ABS) from MathWorks examples as a target CPS to demonstrate multilevel monitoring framework [12]. The ABS system is summarized in the **Fig. 3**. ABS is a safety critical unit in a car that helps prevent the locking of brakes thereby preventing an uncontrollable skid. The slip in a car is calculated based on the wheel rotation speed and actual vehicle speed measured by sensors in the plant (modeled by the vehicle dynamics). This slip value is communicated to the ABS controller through the CAN bus. The ABS controller compares the measured slip and a pre-set threshold slip (chosen so that a slip below this threshold is acceptable for safe operation of the car) and determines if the brake has to be on or off. The brake state (on/off) output determined by the ABS controller is communicated back to the plant through the CAN bus. Some important considerations while designing the monitoring framework are:

### 4.1 Rationale for the monitors used in the ABS controller CPS

Considering the heterogeneous nature of CPS and the attacks that can occur at various levels, we consider three monitors (**Fig. 3**) to detect attacks/faults: Functional monitor M1 at the ABS controller and slip calculation unit, Data monitor M2 at the wheel speed sensor, vehicle speed sensor and brake actuator and Network Monitor M3 at the CAN bus. The rationale for the choice of monitors and their placement are as follows: The data from sensors of dynamic quantities such as vehicle speed or wheel speed can be attacked or corrupted, hence a data monitor (M2) is needed there. At the ABS controller and slip calculator modules, there are various faults that can compromise the functionality of the controller/computational element, hence a functional monitor (M1) is necessary. Finally, by injecting spurious traffic into the CAN bus, genuine data being transmitted between the ABS controller and the plant can be delayed or even distorted.

Therefore, it is necessary to have a network monitor (M3). ABS functionality can be monitored even from the information in the CAN bus. Although, functional monitoring on the CAN bus can offer effective bolt-on solution to existing CPS, it is important to note that the CAN bus has limited observability, all data and functionality we want to monitor may not be available of the CAN bus.

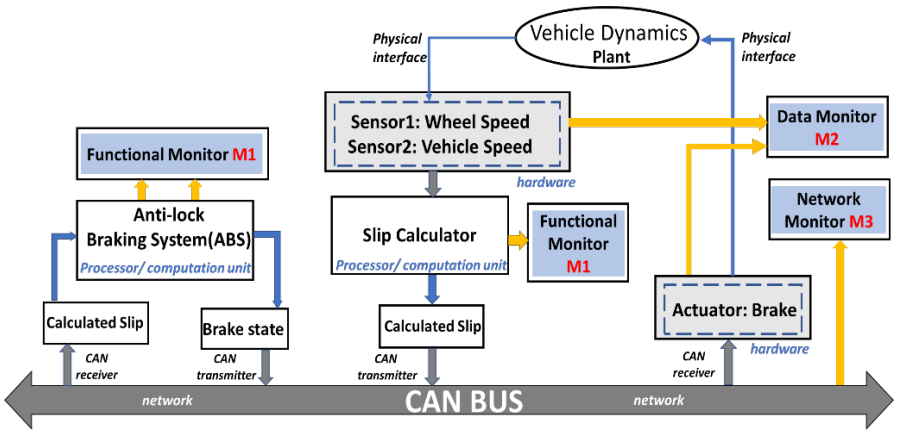


**Fig. 3.** Anti-lock Braking System showing (a) Functional Monitor M1 at the computational units (b) Data monitor M2 at the hardware sensor /actuators level (c) Network monitor M3 at the CAN bus network level.

### 4.2 Monitoring properties expressed using Event Calculus

We define the safety and security properties to be monitored using a formal language called *Event Calculus*. All properties were derived from system level requirements for the specific ABS application. In this example, we focus on application dependent properties, but event calculus is equally expressive for application independent properties. Event calculus is a powerful logical formalism that can conveniently express the effect of events or actions in a CPS in a general way [13]. It is particularly suitable in its ability to express high level functional events as well as low level hardware events. For example, one can express the condition that the temperature of the room increases at a certain rate after a heater in turned on. Formally, in the language of event calculus, switching "on" the heater is an action or an event, that affects the temperature of the room (a fluent) at certain time points. Happens, Initiates, Terminates, HoldsAt and Clipped are the basic event calculus predicates defined in [13]. We use the Happens and HoldsAt predicates to define properties for our system. The semantics of these two predicates are as follows:

- *Happens (α, t)* means that an action or an event α happens at time t.
- *HoldsAt (f, t)* means that the fluent f holds at time t.

To keep the analysis simple, we describe only one property monitored by each of the multilevel monitors (data, network and functional) as an example to explain our framework.

**Property1 verified by Functional Monitor M1:** If the calculated *Slip* is greater than a permissible threshold of $Slip_{safe}$ at time T, then the brake should be off at time T. Here *Slip* is the event and state of the $Brake_{off}$ is the fluent.

$$\text{Happens}(Slip, \text{T}) \wedge \left(Slip > Slip_{safe}\right) \Rightarrow \text{HoldsAt}\left(Brake_{off}, T\right) \quad (1)$$

**Property2 verified by Data Monitor M2** *:* If there is an event on wheel speed *Wheel-Speed_A* at time $T_a$ and another event on wheel speed *WheelSpeed_B* at time $T_b$ where $T_b=T_a+T_d$, then the rate of change of wheel speed $R_w = \frac{(WheelSpeed_B - WheelSpeed_A)}{Td}$ should be less than *Rw_safe* (rate of change of wheel speed for safe operation).

Here $T_d$ is time elapsed between successive wheel speed measurements. $WheelSpeed\_A$ and $WheelSpeed\_B$ are the events and the rate of change of wheel speed being less than the permissible rate of change of wheel speed is the fluent:

$$\text{Happens}(WheelSpeed\_A, \text{Ta}) \wedge \text{Happens}(WheelSpeed\_B, \text{Tb}) \wedge (\text{Tb} = \text{Ta} + \text{Td}) \Rightarrow \text{HoldsAt}\ (\text{Rw} < Rw\_safe, \text{Tb}) \quad (2)$$

**Property3 verified by Network Monitor M3:** If there is a packet arrival in the CAN bus ($Packet_A$) at time $T_a$ and another packet arrival ($Packet_B$) at time $T_b$ then the rate of packet arrival $T_p = T_b - T_a$ should be less than $T_{safe}$ which is the delay in the CAN bus when there is normal traffic for all time T.

Here Tp is time elapsed between successive packet arrivals. Arrival of $Packet_A$ and $Packet_B$ are the events and rate of packet arrival $T_p$ is the fluent:

$$\text{Happens}(Packet_A, \text{Ta}) \wedge \text{Happens}(Packet_B, \text{Tb}) \Rightarrow \text{HoldsAt}\ (\text{Tp} < Tsafe,\ \text{Tb}) \quad (3)$$

The Event Calculus formalisms above combined with Simulink modeling allows designers/modelers to precisely capture monitoring properties.

## 5 Evaluation of Multilevel Monitors

The ABS controller, sensors and the CAN bus were injected with attacks/faults and the efficacy of the monitors in detecting these attacks/faults were evaluated. We used the data injection toolbox in [14] to inject sensor attacks on the model. Fault saboteurs were inserted in the model as explained in [15] at various points in the system. **Fig. 4** shows the saboteurs inserted in the ABS controller. Excessive information packets of higher priority from a malicious node flooding the CAN bus emulated a "Denial of Service" attack. The monitoring conditions were modeled using Simulink assertion verification blocks. We discuss below some examples to demonstrate that (1) there are attacks/fault scenarios that can only be detected if there are localized monitors at each level (data, functional, network) (2) Some attacks/faults may be detected by a monitor at another level (other than the level of its origin), but monitors are nevertheless needed at each

level to locate the origin of the attack/fault in such scenarios. **Table 1** summarizes some of the attacks/faults that were injected in the CPS to demonstrate the need for a multilevel monitoring framework.

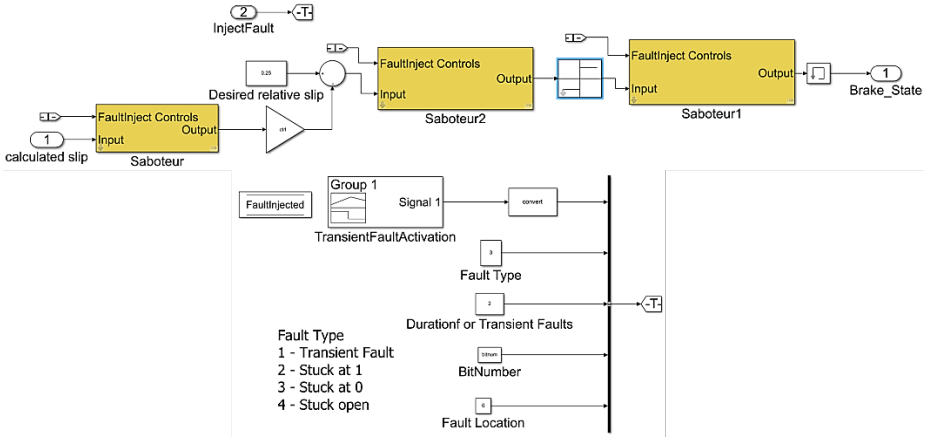


**Fig. 4.** Fault Saboteurs injected in the ABS.

**Table 1.** Attacks/faults injected on the CPS.

| No. | Attack/Fault | Attack location | Monitors that detect |
|---|---|---|---|
| 1. | Stuck-at 0 fault | ABS controller | M1 only |
| 2. | Denial of service attack | CAN bus | M3 only |
| 3. | Sensor measurement injection attack | wheel speed sensor | Attack-1: M2 only<br>Attack-2: M1 and M2 |

### 5.1 Case-1. Attacks/faults needing localized monitors at each level:

Consider the **Fig. 5** where the slip, vehicle speed and wheel rotation speed are plotted as a function of time without the attacks/faults mentioned in the **Table 1**.

When there is no attack/fault, the ABS is able to ensure that the vehicle speed slows down to under 15 m/s at 12 seconds by appropriately releasing the brake whenever the slip exceeds a threshold. In many cases, where there is an attack/fault as shown in **Fig. 6**, **Fig. 7** and **Fig. 8,** the vehicle speed is ~20m/s or higher in 12 seconds (thus rendering the braking ineffective). The ABS controller decides whether the brake should be on/off depending on the slip. When the slip is greater than 0.25 (a threshold value) the brake should be off and when the slip is less than 0.25, the brake should be on.

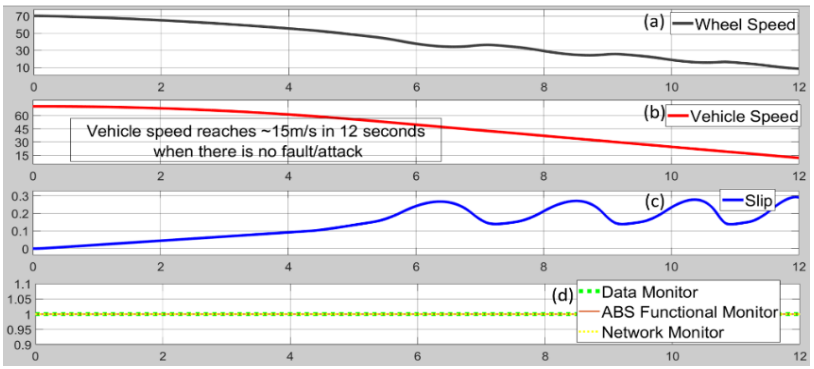


**Fig. 5.** (a)Wheel Speed (b) vehicle speed (c) slip (d) monitor state: when there is no attack/fault on the CPS.

We first consider a fault on the ABS controller which can be critical for the system safety. A "stuck-at zero" fault was injected on the slip at about t=5 seconds and hence the controller never turns the brake off and is always on. Therefore, the property, "the brake is turned off when the slip ("s") is greater than 0.25" is violated. It can be seen in **Fig. 6** that around t=6 seconds, the true slip communicated to the controller exceeds 0.25 and the functional monitor (M1) expects the brake to turn off. However, due to the fault (slip seen by the controller is zero) the controller still keeps the brake on. Hence, the property is violated and fault is detected by the ABS functional monitor. However, since this does not affect the signal transmission through the CAN bus or other sensor properties, the monitors at the network and data levels are unable to detect this. Hence, one specifically needs a functional monitor here to detect the fault.

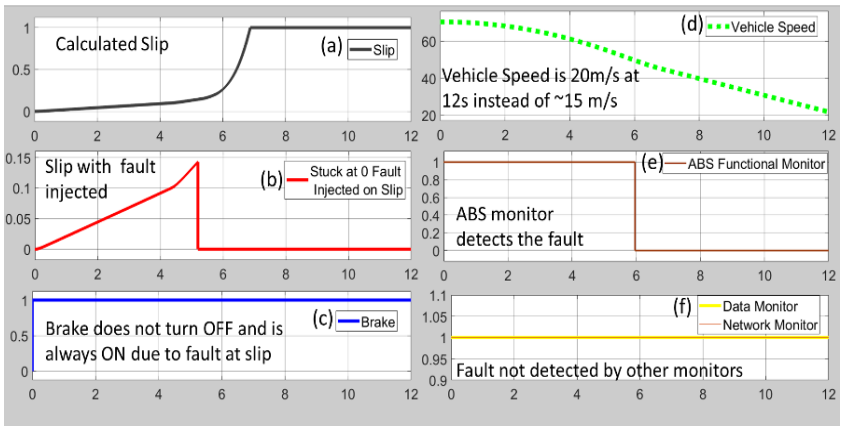


**Fig. 6.** For a stuck at 0 fault on the ABS controller, (a) correct slip calculated (b) slip as seen by the ABS controller due to the fault at its input (c) the brake state which is always "on" as even though the true slip exceeds 0.25, the ABS only sees the slip=0 (d) vehicle speed that is affected by the ABS not correctly functioning (e) the ABS expects the brake to go "off" when slip exceeds 0.25, and thus detects a fault (f) other monitors do not detect this fault.

Likewise, the CAN bus is prone to number of attacks: packet insertion, packet erasure, packet payload modification, to name a few [16]. These lead to Denial of Service (DoS) attack that changes the packet frequency on the CAN bus. Time interval between CAN packets is usually periodic and has a fixed delay. A malicious node can change the time interval between successive packets by injecting extra packets causing delay in the bus.

An attack on the CAN packet frequency was performed by introducing a malicious node that delays the communication to and from the ABS controller. This was not detected by either the functional monitor (M1) at the ABS or the data monitor (M2). The fixed delay for normal traffic was identified and the network monitor (M3) verifies at runtime that the time interval between subsequent packets is within bounds. When the time interval exceeds the normal levels the monitor M3 indicated an attack on the network as shown in **Fig. 7**. When the system has no faults/attacks, the ABS controller receives the slip value, approximately every 0.006 seconds through the CAN bus. However, when there is more than a certain level of network traffic due packet injection by a malicious node, the delay in the CAN bus increases, which is detected by the monitor as shown. Flooding the CAN bus with many packets can lead to huge delay as seen in **Fig. 7** (b) between 11$^{th}$ and 12$^{th}$ second. This affects the braking and the vehicle speed. The vehicle speed was 30m/s instead of 15 m/s during normal conditions with no fault/attack. While we used this approach as a proof of concept, there are alternate ways of monitoring the bus traffic discussed in [16].

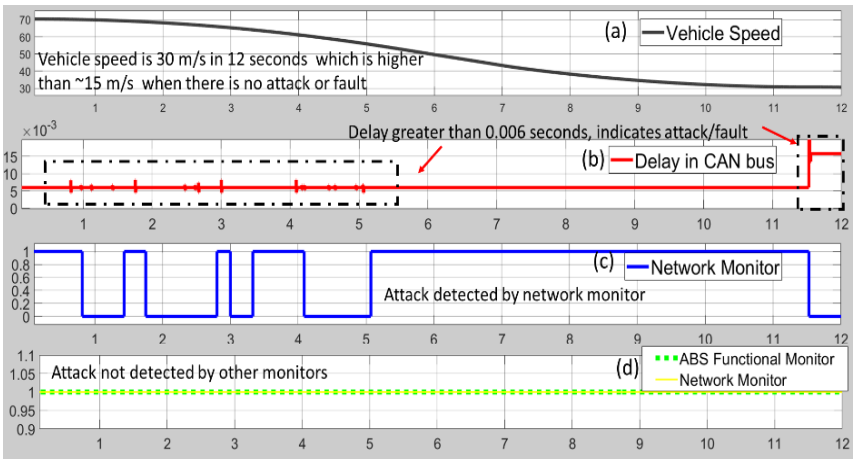


**Fig. 7.** Bus traffic delay detected by Network monitor For a DoS attack, (a) vehicle speed that is affected due to delay in CAN bus (b) delay in CAN bus is greater than 0.006 seconds (c) Network monitor detects the attack (d) all other monitors do not detect the attack.

A sensor attack, “attack-1” on the wheel speed sensor that is detected by the data monitor (M2) is showed in **Fig. 8**. It monitored the safety property “the absolute value of the rate of change of wheel speed should not be greater than $T_w$ rad/sec” where $T_w$ is a threshold rate of change of wheel speed for safe operation. However, *none of the other monitors* were able to detect this attack.

Hence, in all the above cases multilevel monitors are needed as faults/attacks at one level cannot be detected by monitors at the other levels as demonstrated by the above examples. *Hence, we show that having monitors at multiple levels are beneficial (and sometimes required) to detect attacks/faults that span multiple levels and systems.*

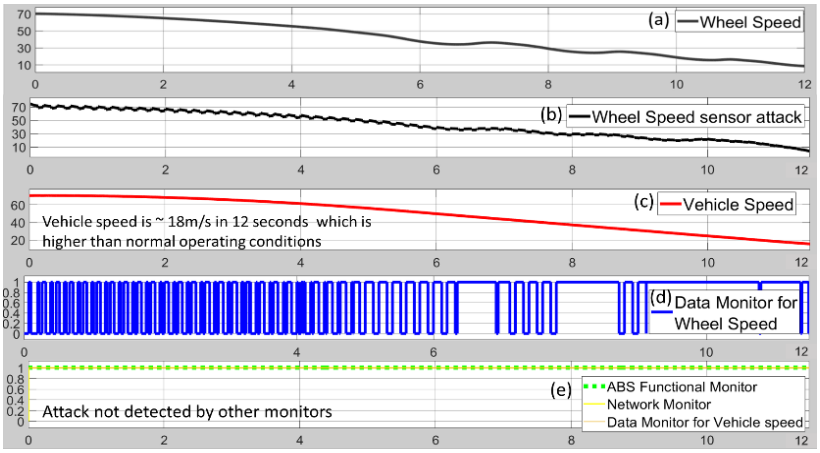


**Fig. 8.** For an attack on wheel speed sensor, (a) Wheel speed when there is no attack/fault (b) wheel speed with an attack (c) vehicle speed affected by attack on the wheel speed sensor (d) Data monitor for wheel speed detects the attack(e) other monitors do not detect this attack.

### 5.2 Case-2. Attacks/faults detected at more than one level but still needing multiple levels to find to location of the attack:

When there is sensor measurement attack (discussed earlier) of a much higher magnitude (attack-2), it could cause the rate of wheel speed to change so drastically that it briefly affects the functional relation between the slip and break state monitored by M1. Hence it is detected by the functional monitor in addition to the wheel speed data monitor as shown in **Fig. 9**. Note that this example has less number of disruptions to the wheel speed and does not significantly change the eventual vehicle speed reached at 12 seconds. However, it is still important to detect any attacks on the CPS.

We argue *both* of these monitors are probably needed, as even though the functional monitor detects this data attack, we cannot be sure where the attack/fault originated if we *only* had one functional monitor. We would use the fact that both the wheel speed data monitor and functional monitor detected this attack to pinpoint it was at the wheel speed sensor; while if only the functional monitor had detected the attack (not the data monitor) we would probably conclude the attack was on the ABS controller.

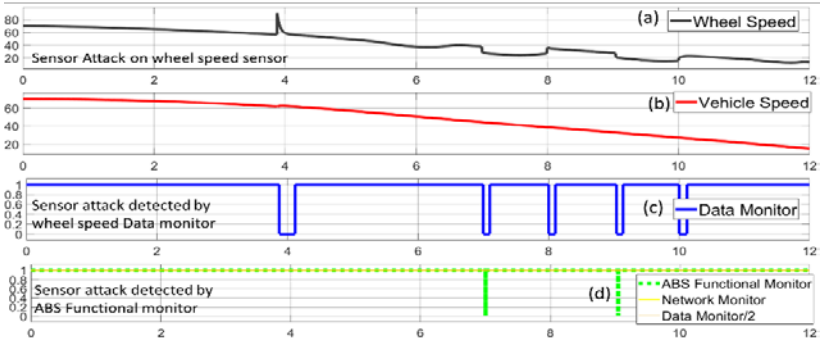


**Fig. 9.** Attack detected by multiple monitors (a) Wheel speed when there is sensor attack (b) vehicle speed not affected significantly by attack on the wheel speed sensor (c) Data monitor for wheel speed detects the attack (d) functional monitor detects this attack.

Another issue to be considered is whether the ABS functionality (M1) and sensor data (M2) can be monitored from the information in the CAN bus. One issue is the CAN bus has limited observability as all data and functionality we want to monitor may not be available of the CAN bus. The other issue is as follows: Suppose the slip and brake state, are available on the CAN bus, we could have implemented the same ABS functional monitor on the slip and Brake ON/OFF state from information in the CAN bus (not shown here) rather than locally as we did earlier. While such a monitor would have detected a fault in the ABS controller action, it would have also been affected by excessive network traffic. So, this monitor alone would not be able to specifically pin point the origin of the attack.

## 6 Conclusion and Future Work

We have developed and implemented a multilevel monitoring framework and demonstrated the need for monitors at multiple levels to detect various attacks/faults for an ABS controller CPS. We showed that existing MBE tools (Simulink) can model and evaluate such monitoring architectures and integrate safety and security considerations early in the design process. Future continuation of this work will; (1) focus on comparisons with other approaches to access the benefits and limitations, (2) further the development of a theory of multilevel monitoring for CPSs to fully characterize its assumptions and impacts. Finally, the generality and scalability of multilevel monitors deployed in diverse CPSs will be better understood by evaluating the resources needed for implementing such monitors.

## References

1. christina-kay.kellerman@nist.gov, “Cyber Physical Systems and Internet of Things Program,” *NIST*, Mar. 09, 2016. https://www.nist.gov/programs-projects/cyber-physical-systems-and-internet-things-program (accessed May 18, 2020).

2. B. Combemale and M. Wimmer, “Towards a Model-Based DevOps for Cyber-Physical Systems,” in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Cham, 2020, pp. 84–94, doi: 10.1007/978-3-030-39306-9_6.
3. A. Kane, “Runtime Monitoring for Safety-Critical Embedded Systems,” Carnegie Mellon University, 2015.
4. S. Gautham, G. Bakirtzis, M. T. Leccadito, R. H. Klenke, and C. R. Elks, “A Multilevel Cybersecurity and Safety Monitor for Embedded Cyber-physical Systems: WIP Abstract,” in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, New York, NY, USA, 2019, pp. 320–321, doi: 10.1145/3302509.3313321.
5. R. Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu, “BusMOP: a Runtime Monitoring Framework for PCI Peripherals,” p. 23.
6. L. Convent, S. Hungerecker, T. Scheffel, M. Schmitz, D. Thoma, and A. Weiss, “Hardware-Based Runtime Verification with Embedded Tracing Units and Stream Processing,” in *Runtime Verification*, vol. 11237, C. Colombo and M. Leucker, Eds. Cham: Springer International Publishing, 2018, pp. 43–63.
7. T. Lu, J. Lin, L. Zhao, Y. Li, and Y. Peng, “A Security Architecture in Cyber-Physical Systems: Security Theories, Analysis, Simulation and Application Fields,” *IJSIA*, vol. 9, no. 7, pp. 1–16, Jul. 2015, doi: 10.14257/ijsia.2015.9.7.01.
8. A. E. Goodloe and L. Pike, “Monitoring Distributed Real-Time Systems: A Survey and Future Directions,” *(NASA/CR-2010-216724)*, p. 49, Jul. 2010.
9. M. W. Whalen, A. Murugesan, S. Rayadurgam, and M. P. E. Heimdahl, “Structuring simulink models for verification and reuse,” in *Proceedings of the 6th International Workshop on Modeling in Software Engineering - MiSE 2014*, Hyderabad, India, 2014, pp. 19–24, doi: 10.1145/2593770.2593776.
10. E. A. Lee and S. A. Seshia, *Introduction to embedded systems: a cyber-physical systems approach*, Second edition. Cambridge, Massachuetts: MIT Press, 2017.
11. A. P. Fournaris, A. Komninos, A. S. Lalos, A. P. Kalogeras, C. Koulamas, and D. Serpanos, “Design and Run-Time Aspects of Secure Cyber-Physical Systems,” in *Security and Quality in Cyber-Physical Systems Engineering*, S. Biffl, M. Eckhart, A. Lüder, and E. Weippl, Eds. Cham: Springer International Publishing, 2019, pp. 357–382.
12. “Effects of Communication Delays on an ABS Control System - MATLAB & Simulink.” https://www.mathworks.com/help/simevents/examples/effects-of-communication-delays-on-an-abs-control-system.html (accessed May 18, 2020).
13. M. Shanahan, “The Event Calculus Explained,” in *Artificial Intelligence Today*, vol. 1600, M. J. Wooldridge and M. Veloso, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 409–430.
14. S. Potluri, C. Diedrich, S. R. Roy Nanduru, and K. Vasamshetty, “Development of Injection Attacks Toolbox in MATLAB/Simulink for Attacks Simulation in Industrial Control System Applications,” in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Jul. 2019, vol. 1, pp. 1192–1198, doi: 10.1109/INDIN41052.2019.8972171.
15. A. V. Jayakumar, “Systematic Model-based Design Assurance and Property-based Fault Injection for Safety Critical Digital Systems,” *Theses and Dissertations*, Jan. 2020, [Online]. Available: https://scholarscompass.vcu.edu/etd/6239.
16. S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar, “Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review,” *J Wireless Com Network*, vol. 2019, no. 1, p. 184, Dec. 2019, doi: 10.1186/s13638-019-1484-3.