Butterfly Attack: Adversarial Manipulation of Temporal Properties of Cyber-Physical Systems

Rouhollah Mahfouzi
Embedded Systems Laboratory
Linköping University (LiU)
Linköping, Sweden
rouhollah.mahfouzi@liu.se

Amir Aminifar
Embedded Systems Laboratory
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland
amir.aminifar@epfl.ch

Soheil Samii
General Motors R&D (GM)
Warren, MI, USA
Embedded Systems Laboratory
Linköping University (LiU)
Linköping, Sweden
soheil.samii@{gm.com,liu.se}

Mathias Payer
HexHive Systems Security Group
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland
mathias.payer@epfl.ch

Petru Eles
Embedded Systems Laboratory
Linköping University (LiU)
Linköping, Sweden
petru.eles@liu.se

Zebo Peng
Embedded Systems Laboratory
Linköping University (LiU)
Linköping, Sweden
zebo.peng@liu.se

***Abstract*—Increasing internet connectivity poses an existential threat for cyber-physical systems. Securing these safety-critical systems becomes an important challenge. Cyber-physical systems often comprise several control applications that are implemented on shared platforms where both high and low criticality tasks execute together (to reduce cost). Such resource sharing may lead to complex timing behaviors and, in turn, counter-intuitive timing anomalies that can be exploited by adversaries to destabilize a critical control system, resulting in irreversible consequences. We introduce the butterfly attack, a new attack scenario against cyber-physical systems that carefully exploits the sensitivity of control applications with respect to the implementation on the underlying execution platforms. We illustrate the possibility of such attacks using two case-studies from the automotive and avionic domains.**

***Index Terms*—security attack, real-time scheduling, control applications, resource sharing, stability analysis, time delay, latency, jitter.**

I. Introduction

Cyber-physical systems are shifting towards integrated architectures, as observed, e.g., in the automotive domain [1]. As a result, the majority of control applications in such domains are implemented as software tasks on commodity platforms, sharing the CPU with other tasks. If not carefully taken into account, this resource sharing leads to complex timing behaviors, which may jeopardize the stability of the control applications [2]–[4].

Cyber-physical systems [5] tightly interact with secondary networks and components, giving an adversary ample opportunities to provide malicious input, potentially compromising the safety of the system. Unfortunately, many such systems comprise crucial physical components, e.g., adaptive cruise control or engine control in automotive systems that allow deep intervention in the driving of a vehicle [6]. Therefore, ignoring the security aspects in cyber-physical systems will have severe consequences.

By the interconnection of the physical systems to the cyber (processing) elements, the notion of physical time is introduced in today's embedded systems. Temporal attacks are a new class of attacks where the correct behavior of the system is targeted through manipulation of its temporal properties. The very first example could be execution overruns of an application that can lead to deadline misses in other applications. Such deadline misses, in turn, may have catastrophic consequences in safety-critical applications. Following best practices, overruns are considered in the design phase and appropriate measures are planned in such cases. However, as we discuss later, temporal attacks are by no means limited to overruns. In fact, there are non-trivial cases where the safety of a system is jeopardized if a non-critical task simply stops executing. In control applications, this may lead to longer jitter in execution of the critical control task and instability of the physical plant. Note that although the non-critical application does not violate its limit on resource usage, the physical plant associated with the control application is compromised. Similarly and counter-intuitively, if a task decides to switch to a lower priority level or longer period, i.e., giving up certain privileges, the stability of other control applications can be compromised.

The performance and stability of physical plants are related to the computational resources provided to their corresponding control tasks, which are running on the shared platform [7]–[9]. This has been extensively studied in the literature related to control under co-designing scheduling constraints [10]–[42]. Nevertheless, it is widely believed that giving more resources (e.g., amount of available processor) to a controller leads to better control quality and only reducing the amount of resources represents a threat (monotonicity property). In the Vestal model [43], widely used for the analysis of (mixed) critical systems, the implicit assumption is that exceeding a certain worst-case execution/computation time is the main threat to be considered. Thus, if a job attempts to consume more time than allowed in a certain mode (by e.g., attempting to run for a longer execution time, or more frequently) the criticality mode of the system changes [44]. If this mode change is realized, it is assumed that the necessary isolation/protection of high criticality tasks, in the time domain, has been achieved. Similarly, in the real-time systems area, it is assumed that providing guarantees for the case of a given upper bound on execution time and lower bound on inter-arrival time is sufficient and safe. However, as we pointed out previously, these assumptions are not sufficient to avoid security threats in the timing domain.

In this article, we consider a new attack scenario for cyber-physical systems, in which the adversary can interfere with an entry task[1] running on the shared platform (this task is not needed to be associated with the targeted critical application). The attacker, however, does not explicitly interfere with the execution of the targeted applications (e.g., by manipulation of registers and pointers, interfering with sensors or actuators, or similar to a replay/stealthy/denial-of-service attack). In addition, as a result of the attack, the entry task does not consume more resources (processor bandwidth/time) than the allocated ones (e.g., overruns are categorized as denial-of-service attack). The attacker will indirectly manipulate the usage of the provided resources (by providing certain inputs to a certain entry task running on the platform) in such a way that the stability of the plants associated with other (targeted) tasks is jeopardized. The attack scenario is illustrated in Figure 1.

Observe that, as opposed to most common attack models, here, the adversary does *not actively interfere* with the correct execution of the targeted critical tasks/applications. The attacker will manipulate the execution of another possibly low criticality entry task (not the targeted application), by providing certain legitimate inputs to it. As a result of this manipulation, the entry task will consume less (not more) resources (time) than possibly allowed.

Among temporal attacks, the attack scenario presented in this article exploits the observation that complex cyber-physical systems are so entangled that manipulating one entity of the system can tamper with the functionality of other entities, hence suggesting chaotic behaviors and the name Butterfly effect and attack.

The rest of the paper is organized as follows. Section II introduces the necessary background for this paper. Section III describes our assumption on the environment and capabilities of the adversary. In Section IV, the Butterfly attack is defined and two of its main characteristics are illustrated. An example is provided in Section V to elaborate on the Butterfly attack. Section VI and Section VII illustrate two real-world case studies where the Butterfly results in unintended potentially devastating behavior. The mitigation and detection solutions are discussed in Section VIII. Finally, the related work is discussed in Section IX and Section X concludes the paper.

[1]The lower criticality less protected task that interacts with the outside world. The adversary *indirectly* attacks the targeted critical control application through this entry task.

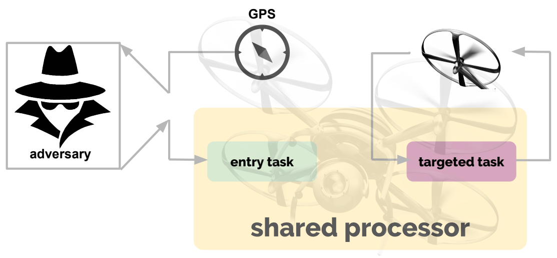


Fig. 1: Illustration of the attack scenario. As a result of intervention (e.g., jamming GPS signals) the entry task consumes less resources and this might lead to instability of the targeted task.

II. Background

In this paper, we assume a uniprocessor platform Γ which is shared among a set of control applications $\mathbf{\Lambda}$. Each control application includes a control task and a physical plant. The set of all control tasks and all physical plants is denoted by $\mathbb{T}$ and $\mathbb{P}$, respectively.

A. Task Model

On the shared processor platform Γ, a task related to control application $\Lambda_i \in \mathbf{\Lambda}$ is denoted by $\tau_i \in \mathbb{T}$. Tasks have different criticality levels. The criticality level of task τ_i is denoted by $\chi(\tau_i)$. Tasks are periodic and real-time and can be preempted. The computation time (execution time), sampling period, and priority of task $\tau_i \in \mathbb{T}$ are denoted by c_i, h_i, and ρ_i, respectively. Each instance of a task (a job) might have different computation time depending on the input. The computation time c_i is bounded from below by c_i^{b} and from above by c_i^{w}. The set of higher priority tasks for task $\tau_i \in \mathbb{T}$ is denoted by $hp\left(\tau_i\right) \subseteq \mathbb{T}$. Task $\tau_i \in \mathbb{T}$ has a higher priority than task $\tau_j \in \mathbb{T}$ if $\rho_i > \rho_j$.

We assume the tasks are independent, deadlines are implicit, and fixed-priority preemptive scheduling is applied. While we assume fixed-priority scheduling for the discussion in this section, the Butterfly attack is equally applicable in the case of other scheduling policies.

B. Plant Model

We model plant $P_i \in \mathbb{P}$, related to control application $\Lambda_i \in \mathbf{\Lambda}$, and controlled by control task $\tau_i \in \mathbb{T}$, by a continuous-time system of differential equations [7],

$$\dot{\boldsymbol{x}}_i = \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{u}_i, \tag{1}$$

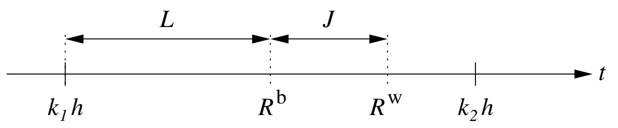


Fig. 2: Graphical interpretation of the latency (L) and response-time jitter (J). R^{w} and R^{b} denote the worst-case and best-case response time and h is the period of the controller.

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and the control signal, respectively. The sampling is done periodically, and the control signal is updated with some delay which depends on real-time task scheduling.

In this shared processor model, tasks need to be scheduled in a way that they meet their deadlines and the physical plant is kept stable.

C. Stability Analysis

Stability of a control plant is defined mathematically based on the plant model [7]. In order to measure the stability of a controller associated with plant P_i, we use a standard quadratic cost [7]:

$$\lim_{T\to\infty}\frac{1}{T}\mathbb{E}\left\{\int_0^T \begin{bmatrix}\boldsymbol{x}_i\\ \boldsymbol{u}_i\end{bmatrix}^{\mathsf{T}} \boldsymbol{Q}_i \begin{bmatrix}\boldsymbol{x}_i\\ \boldsymbol{u}_i\end{bmatrix} dt\right\}. \tag{2}$$

The weight matrix $\boldsymbol{Q}_i$ is given by the designer, and is a positive semi-definite matrix with weights that determine how important each of the states or control inputs are in the final control cost, relative to others ($\mathbb{E}\{\cdot\}$ denotes the expected value of a stochastic variable). An infinite value of the above control cost indicates that the control application is not stable [45]. For example, a quadcopter is stable if it is kept in a specific angle (equilibrium). If the angle of the quadcopter diverges from the equilibrium and the control cost for the controller to return it back is infinity, then the quadcopter is unstable.

The stability of control applications depends on the characteristics of the delays introduced in the control feedback loop. Consequently, latency and jitter are two important metrics when analyzing the stability of a periodic control application. Increased latency and jitter in providing the output of a control application will lead to poor control performance (increase in control cost) and can even jeopardize the stability of the physical plant. When control tasks are running on a shared platform, the tasks may encounter varying delays in providing the output to the actuator in different iterations. Therefore, the physical plant, which is actuated after the controller task finishes executing, would experience unexpected varying delays. Hence, it is important to consider the latency and jitter introduced during the implementation of cyber-physical systems in order to be able to guarantee the stability of the physical plant. Considering a control application Λ_i, the latency L_i is defined as the constant part of the delay experienced by the actuator and the worst-case jitter J_i is defined as the variation in the delay. The two metrics are defined as follows (see Figure 2),

$$\begin{aligned} L_i &= R_i^{\mathrm{b}},\\ J_i &= R_i^{\mathrm{w}} - R_i^{\mathrm{b}}, \end{aligned} \tag{3}$$

where R_i^{w} and R_i^{b} denote the worst-case and best-case delay (response time) experienced by the actuator.

In the following, we elaborate the dependency between the latency and jitter experienced by the control application and its stability. We use the Jitter Margin tool [8] to capture this dependency. For a given plant model and latency, the Jitter Margin toolbox computes the *jitter margin* (similar to the *phase margin* and *gain margin* concepts of control theory). That is, the Jitter Margin toolbox provides the stability curve that determines the maximum tolerable jitter J_i, based on the experienced latency L_i.

The solid curve in Figure 3 is an example of the *stability curves* generated by the Jitter Margin toolbox, where the area below the curve is the stable area. This solid curve is generated for a DC servo process with transfer function $\frac{1000}{s^2+s}$ and a discrete-time Linear-Quadratic-Gaussian (LQG) controller, with a sampling period of 6 ms.

The stability curve can safely be approximated by a linear function of the latency and worst-case jitter [9], see dashed line in Figure 3. The *stability condition*, hence, can be formulated as,

$$L_i + \alpha_i \cdot J_i \le \beta_i, \tag{4}$$

where $\alpha_i \ge 1$ and $\beta_i \ge 0$ are constant coefficients.[2]

D. Worst-Case and Best-Case Response Time

In this section, we give a brief overview on computing the worst-case, and best-case response time. Considering the system model mentioned in Section II-A the exact worst-case response time of a task τ_i can be computed by the following equation [46],

$$R_i^{\mathrm{w}} = c_i^{\mathrm{w}} + \sum_{\tau_j\in hp(\tau_i)} \left\lceil \frac{R_i^{\mathrm{w}}}{h_j} \right\rceil c_j^{\mathrm{w}}. \tag{5}$$

Similarly, the exact best-case response time of a task τ_i is given by the following equation [47],

$$R_i^{\mathrm{b}} = c_i^{\mathrm{b}} + \sum_{\tau_j\in hp(\tau_i)} \left\lceil \frac{R_i^{\mathrm{b}}}{h_j} \right\rceil c_j^{\mathrm{b}}. \tag{6}$$

The above equations are used to calculate the worst-case jitter (see Equation 3) and analyze the stability of the control applications (Equation 4) in a provided schedule. Therefore, by combining Equations 3, 4, 5, and 6, the

[2]We consider the simple linear stability condition for simplifying the discussion in the rest of the paper. In practice, a piecewise linear approximation can be used in order to achieve better accuracy. However, the Butterfly attack is even applicable in the cases that the stability curve cannot be approximated by a piecewise linear function.

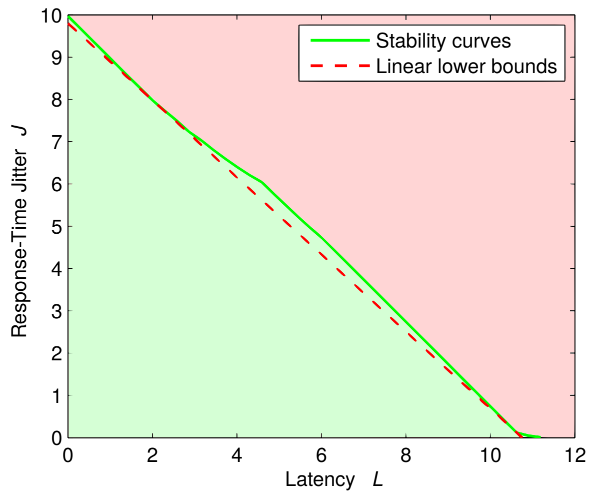


Fig. 3: The stability curve generated by Jitter Margin and the linear lower bound (the area below the curve is the stable area). The curve is generated for a DC servo process with transfer function $\frac{1000}{s^2+s}$ and a discrete-time Linear-Quadratic-Gaussian (LQG) controller, with a sampling period of 6 ms.

stability condition for a task τ_i can be written as follows,

$$(1-\alpha_i)\cdot\left(c_i^{\mathrm{b}}+\sum_{\tau_j\in hp(\tau_i)}\left\lceil\frac{R_i^{\mathrm{b}}}{h_j}\right\rceil c_j^{\mathrm{b}}\right)+ \alpha_i\cdot\left(c_i^{\mathrm{w}}+\sum_{\tau_j\in hp(\tau_i)}\left\lceil\frac{R_i^{\mathrm{w}}}{h_j}\right\rceil c_j^{\mathrm{w}}\right)\leq\beta_i. \tag{7}$$

Accordingly, the stability of task τ_i depends on computation time, period, and priority of its own and other tasks with higher priority than task τ_i. This dependence will be exploited in the next section to describe a new attack on real-time control systems.

III. Threat model

We consider the following environment in the context of a Butterfly attack:

- A processing platform Γ shared among a set of tasks $\mathbb{T}$;
- An arbitration mechanism based on which the tasks in task set $\mathbb{T}$ are scheduled at runtime;
- A safety-critical and real-time controller task $\tau_i \in \mathbb{T}$ that is responsible for stabilizing a physical plant P_i.

We assume the following adversary in this article:

- The adversary may indirectly change the timing behavior of non-critical task $\tau_j \in \mathbb{T}$ (entry task) that shares the platform with the safety-critical controller task $\tau_i \in \mathbb{T}$ (targeted task).
- The adversary is aware of the non-critical (less protected) task τ_j (entry task) running on the shared platform with the safety-critical control task τ_i (targeted task).

The attack scenario presented here is considered to be *indirect* in the following sense:

- First, the adversary does not need to directly manipulate the shared platform, the scheduling mechanism, or the tasks in the task set $\mathbb{T}$ running on the shared platform, neither the critical tasks, nor the non-critical tasks.
- At the same time, the adversary only indirectly interferes with the execution of non-critical tasks. That is, the adversary uses simple techniques (e.g., jamming the input signal, providing certain inputs to the task, or delay/advance the incoming messages) to indirectly reduce the execution time, delay, or prevent the execution of a non-critical task τ_j. This is possible because the execution time (as well as the execution period) of a task depends on the values of its inputs as well as the order and timing of its inputs.

We assume that the controller task τ_i is protected from attacks and the scheduler allocates sufficient amount of resources to it. The goal of the adversary is to destabilize the physical plant P_i, by indirectly manipulating the temporal proprieties of the safety-critical task τ_i, via lower criticality less protected task τ_j. The interference with task τ_j may also be indirect, e.g., jamming the input (e.g., GPS) signal, providing certain inputs to the task, or delay/advance the incoming messages to manipulate its temporal properties. Such interference techniques are well studied and have been performed in various real life contexts [48]–[54].

IV. Butterfly Attack

In this section, we define the notion of the Butterfly attack. The Butterfly attack demonstrates that, contrary to popular belief [43], allocating more resources to the task under analysis, may jeopardize the stability of control applications and can be exploited by the adversaries. In short: over provisioning does not guarantee non interference.

The Butterfly attack renders a task unstable by indirectly manipulating the time delay it experiences. As we discussed in Section II-C, for a specific latency, a control application is resilient up to a certain amount of jitter. If an attacker changes the timing parameters of one or several tasks (entry tasks) such that it causes an increase in the jitter of the task under attack (targeted task), then the control application associated to the targeted task may become unstable. From Equation 7, we can see that jitter and, hence, stability is related to best-case and worst-case computation time, periods, and priorities of all the tasks running on the shared platform. Therefore, changing any of these parameters can increase jitter and lead to instability of control applications.

Let us assume that a set of control applications $\mathbf{\Lambda}$ share a resource, e.g., processing or communication infrastructure. That is, the set of tasks $\mathbb{T}$ run on this shared platform.

Definition 1: Temporal properties of control application $\Lambda_k \in \mathbf{\Lambda}$ capture the delay patterns experienced by task τ_k, which influence the control performance and stability of Λ_k. The temporal properties of task τ_k depend on the computation time (execution time), period, priority, release time, and offset of task τ_k, as well as those of other tasks competing on the same platform for shared resources.

We consider tasks $\tau_i \in \mathbb{T}$ and $\tau_j \in \mathbb{T}$. We assume task τ_j is less critical (hence less protected) than τ_i i.e., $\chi(\tau_j) \leq \chi(\tau_i)$. The Butterfly attack on control application Λ_i is defined as follows.

Definition 2: Indirect manipulation of temporal properties of a less critical task τ_j to modify the schedule of a more critical task τ_i such that the application Λ_i (τ_i relates to application Λ_i) is out of its expected behaviour is referred to as Butterfly attack.

Such temporal manipulations may be adopted by adversaries to deteriorate the performance of control applications or even destabilize the physical plant P_i controlled by task τ_i.

The Butterfly attack exploits two important characteristics of the control tasks that are utilizing a shared platform. These two characteristics make the Butterfly attack undetectable and effective in many real-life control applications (see Section VI and Section VII):

1) **Inter-dependency:** The computation time, priority and period of tasks are considered independent. Nevertheless, since the tasks run on a shared platform, the temporal properties and, in turn, stability of the tasks depend on each other. In fact, changing the parameters (computation time, period, priority) of one task may alter the temporal properties of other tasks. Therefore, an attacker might interfere with the parameters of one (less critical/protected) task, in order to manipulate the temporal properties of another (more critical/protected) task and make it unstable.
2) **Non-monotonicity:** Temporal properties (and hence stability) of a physical plant is related to the resources provided to its control task. Decreasing the amount of resources for a task will decrease the performance of the controller and may jeopardize the stability of its physical plant. However, the relation between the real-time parameters and temporal properties of tasks is not necessarily monotonic. Specifically, it can be shown that (see Section V) increasing resources (e.g., processor share) available to a task may increase the jitter and, hence, reduce the performance of the controller or even jeopardize the stability of the physical plant. Therefore, counter-intuitively, an attacker can destabilize a task by increasing the available resources to that task.

Note that because of the two aforementioned characteristics of the Butterfly attack, it is not trivial to detect and mitigate it. The system designers try to keep a critical task stable by protecting *only that task* from attackers and making sure that *the task has sufficient resources* to be executed [43]. The Butterfly attack demonstrates that this is not enough. First of all, the attacker renders a control application unstable without interfering the timing parameters of the task under attack but other non-critical (potentially less protected) tasks. Second, the attacker does not necessarily need to reduce the amount of resources allocated for a task in order to make it unstable but it can make a task unstable by increasing the resources available to the task under attack (by, e.g., decreasing the amount of resources reserved to other tasks).

In the next sections, we will demonstrate how the Butterfly attack can be applied, based on the inter-dependency and non-monotonicity present in complex real-time control and cyber-physical systems.

V. Attack Example

In the previous sections, we showed that the stability of control applications depends on the temporal properties experienced, particularly in terms of the amount of jitter. In this section, we use an example to demonstrate that, intuitively allocating a larger processor share to a task, does not necessarily lead to better stability results, but on the contrary, can deteriorate their performance, due to the inter-dependency and non-monotonicity of jitter.

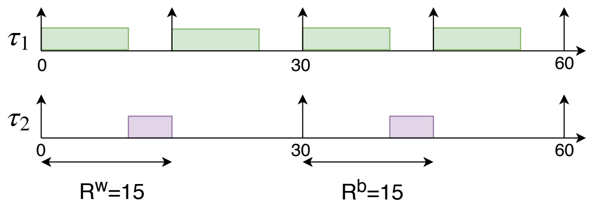
Let us focus on an example, where we assume an implicit deadline (deadline equal to the period) and a constant computation time for each task τ_i ($c_i = c_i^{\mathrm{w}} = c_i^{\mathrm{b}}$). Then, each task τ_i can be modeled by the tuple (ρ_i, c_i, h_i) of priority, computation time, and period, respectively. The sampling period h_i and the priority are parameters set at design time. The computation time can be obtained using execution/computation time analysis tools, such as the aiT tool [55], [56], that take into consideration also the specific properties of the hardware implementation platforms.

In our example, the original task set has two tasks $\mathbb{T} = \{\tau_1, \tau_2\}$, where $\tau_1 = (H, 10, 15)$, $\tau_2 = (L, 5, 30)$. Figure 4(a) shows the original task set and the corresponding schedule. Priorities are assigned according to Rate Monotonic (RM) scheduling. The upward arrows show the release of each job.

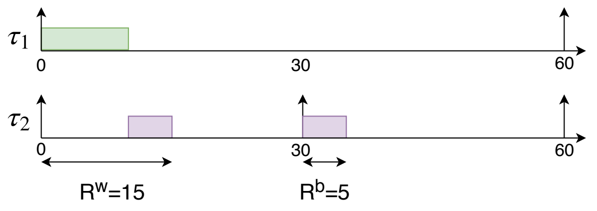
We assume that task τ_2 is the control task of interest (targeted task). According to Figure 4(a), for this task set, the worst-case response time is $R_2^{\mathrm{w}} = 15$ time units and the best-case response time is also $R_2^{\mathrm{b}} = 15$ time units. This means that the response-time jitter is $J_2 = R_2^{\mathrm{w}} - R_2^{\mathrm{b}} = 15 - 15 = 0$ time units.

Let us increase the sampling period of the higher priority task τ_1 to $h_1 = 60$ time units. As it is shown in Figure 4(b), compared to the original task set, the worst-case response time is the same as before, i.e., $R_2^{\mathrm{w}} = 15$ time units, but the best-case response time decreases by 10 time units to $R_2^{\mathrm{b}} = 5$ time units. This leads to an increase in jitter, i.e., $J_2 = R_2^{\mathrm{w}} - R_2^{\mathrm{b}} = 15 - 5 = 10$ time units.

This means that increasing the sampling period of the higher priority tasks can lead to an increase in the jitter that the task under analysis experiences. This indicates the inter-dependency and non-monotonicity of jitter with respect to the sampling period of the higher priority tasks. Finally, this example is also valid for a non-preemptive communication infrastructure, e.g., CAN bus.



(a) Before increasing the period of task τ_1.



(b) After increasing the period of task τ_1 from 15 to 60.

Fig. 4: Schedule of two tasks using a shared platform. The green task has a higher priority and the properties of the tasks are $\tau_1 = (H, 10, 15)$, $\tau_2 = (L, 5, 30)$.

To conclude this section, counter-intuitively, decreasing the high-priority interference does not necessarily lead to better response-time jitter values for the task under analysis. Therefore, the inter-dependency and non-monotonicity of jitter may lead to instability of the plant associated with the task under analysis, if not properly taken into account, and can be exploited by the adversaries to carry out attacks.

VI. Case Study 1: Automotive

Attacks against modern automobiles have gained a lot of attention [57], [58], mainly due to the critical nature of such systems. Recently, Miller and Valasek [58] demonstrated a remote attack against commercial automobiles [51]. The goal of this attack is to remotely control critical tasks, e.g., control steering wheel, engine, transmission, and braking system, which communicate over the CAN bus. This attack exploits vulnerabilities in the entertainment system of a car through a Wi-Fi connection. However, the multimedia system is not connected to the CAN bus directly. The isolation, or so-called air-gap, between the entertainment and critical functionality in vehicles is envisioned to reduce the attack possibilities. Nevertheless, while the multimedia system cannot directly connect to the CAN bus, it can still communicate with another component which is connected to CAN bus, i.e., the V850 controller. The attackers could discover an opportunity to change the firmware of the V850 controller [59] and take control over it. The V850 processor is responsible for interacting with the CAN buses. Once the attacker has access to the V850 controller, it is possible to temporarily stop a message by forcing a collision between the target Electronic Control Unit (ECU) and the attacker-controlled ECU.

The Miller and Valasek [58] attack shows that, in practice, it is possible to take-over CAN communication channels inside a car remotely. Manipulating the critical messages and applications directly can be easily/quickly identified and be counteracted by the existing intrusion detection/mitigation mechanisms [60]–[63]. However, control applications are often (to a certain extent) robust to disturbance, i.e., can tolerate a few packet manipulation and packet drops [64]. As a result, the intrusion detection mechanisms might successfully switch to mitigation mechanisms to ensure stability of the control applications.

In the Butterfly attack, however, we assume that an adversary is more restricted and, at the same time, more stealthy. We assume that the adversary can only temporarily prevent some non-critical CAN messages from using the CAN bus. The Butterfly attack provides the opportunity to target high-critical applications, through manipulating low-critical applications, which often represent less sensitive cases for intrusion detection mechanisms. The attack shows that an adversary may delay messages of a non-critical and less sensitive application, that leads to increase in period of its task, to increase the jitter experienced by a high-criticality task and application (similar to Section V). As a result of such an attack, the high-criticality control application may become unstable due to this increase in jitter, as discussed in Section II-C.

In the following, we show the effectiveness of the Butterfly attack by launching it on a DC motor control application functioning inside an electric vehicle. Our case study considers a subsystem consisting of a highly critical application that controls a DC motor in a battery electric vehicle. On the same ECU hosting the DC motor control application also runs propulsion control software, cruise control, and a high-precision vehicle positioning algorithm. These control applications all receive inputs from multiple sensors and ECUs communicating over a shared CAN bus. These include GPS position from a telematics ECU, wheel speed sensors, and an inertial measurement unit for pitch, roll, and yaw rate. Hosting a large variety of functions on a single ECU is a current trend in automotive industry, often referred to as ECU consolidation or up-integration [65], [66]. While cruise control and vehicle positioning are important functions, these are less critical than the control of the DC motor in a vehicle. We, thus, consider that the CAN messages of the DC motor control are protected, and the adversary is only able to block CAN messages used by the less critical tasks.

Let us consider a DC motor setting proposed in [67]. The mathematical model of the plant is provided as in the following differential equation,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{8}$$

where $\boldsymbol{x}$ and $\boldsymbol{u}$ are the state of the DC motor (rotational speed of the rotor and the armature current) and input signal

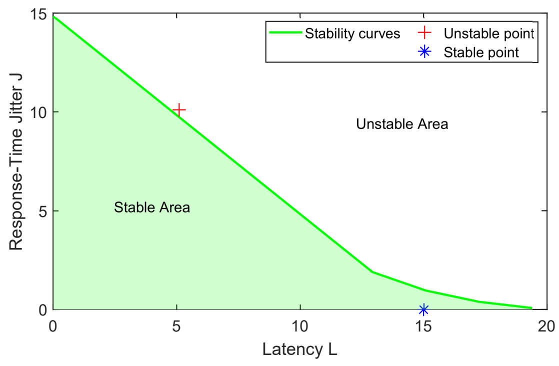


Fig. 5: The stability curve generated by Jitter Margin (the area below the curve is the stable area). The shaded green region is the area in which the plant is guaranteed to be stable. The blue star (latency $L = 15$ and jitter $J = 0$) shows a stable point and the red plus (latency $L = 5$ and jitter $J = 10$) shows an unstable point.

(armature voltage). The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are extracted based on the physics of the DC motor as follows,

$$\boldsymbol{A} = \begin{bmatrix} -11 & 0 \\ 1 & 0 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 32 \\ 0 \end{bmatrix}. \tag{9}$$

Considering this model and using the jitter margin toolbox, the stability curve is depicted in Figure 5 and therefore the stability condition for the plant is as follows,

$$L + 1.13 \cdot J \leq 15.06. \tag{10}$$

We assume that the sensor is periodically sending messages over CAN to the DC motor controller. The controller task is receiving its input from the CAN bus and a job is activated after receiving the input. The CAN bus is shared among several other control applications, i.e., propulsion control software, cruise control, and a high-precision vehicle positioning algorithm. Let us focus only on the *high-precision vehicle positioning* and *DC motor controller* tasks using the CAN bus to communicate. The two applications have the following parameters,

$$\begin{aligned} \tau_1 : \quad & \rho_1 = H, c_1 = 10, h_1 = 15, \\ \tau_2 : \quad & \rho_2 = L, c_2 = 5, h_2 = 30, \end{aligned} \tag{11}$$

where task τ_2 is the task that is running the *DC motor controller* to control the rotational speed of its rotor and is the target task for the adversary, while task τ_1 is the *high-precision vehicle positioning* task with a lower criticality. CAN messages are scheduled using the non-preemptive fixed-priority scheduling algorithm. Priorities are assigned according to RM scheduling.

For simulating the shared platform, we use the True-time toolbox [68] to provide the scheduling of the tasks. TrueTime is a MATLAB/Simulink-based simulator for real-time control systems, which facilitates co-simulation of controller task execution in real-time kernels. If the fixed-priority scheduling policy is adopted, then the tasks execute according to the schedule represented in Figure 4(a). As a result, the latency and jitter for the DC motor controller task would be 15ms and 0ms, respectively. This latency and jitter is shown by a blue star in Figure 5 and the point lays in the stable area. Moreover, according to the stability condition in Equation 10, the DC motor is guaranteed to be stable with this schedule and this is also observed in our experiment. In our experiment, using our control–scheduling co-simulation framework, we change the desired rotor speed at time 1s and the rotor speed converges to the desired speed after some oscillation (see Figure 6(a)) and the control cost is kept finite (see Figure 6(b)), hence the system remains stable.

In order to perform the Butterfly attack, we run the same simulation and change the desired rotor speed at time 1s. However, this time, the adversary increases the period of the messages for task τ_1, i.e., *high-precision vehicle positioning task*, from 15ms to 60ms (e.g., by temporarily interrupting CAN messages from the GPS sensor). Note that, by increasing the period of the messages triggering τ_1, we increase the activation period of the task and, thus, increase the available processor share for the *DC motor task*. Intuitively, this should lead to better control performance in the DC motor. However, Figure 4(b) shows that the latency and jitter of τ_2 is 5ms and 10ms, respectively. This latency and jitter is shown by a red plus in Figure 5 and the point lays in the unstable area. Moreover, according to Equation 10, the DC motor is not guaranteed to be stable with this latency and jitter,

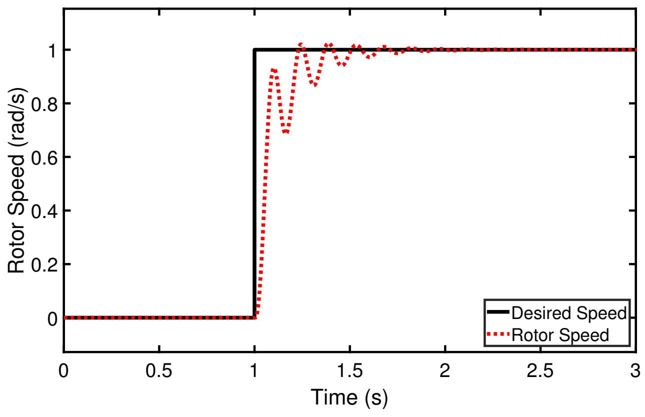
$$L + 1.13 \cdot J \nleq 15.06. \tag{12}$$

This is also confirmed in our control–scheduling co-simulation framework. Figure 7(a) shows that the controller is not able to converge the rotor speed into the desired speed and the control cost goes to infinity (see Figure 7(b)), which indicates that the plant becomes unstable.

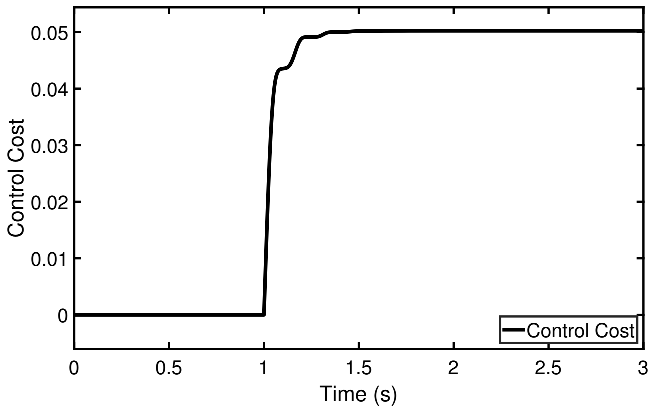
VII. Case Study 2: Unmanned Aerial Vehicles (UAV)

Unmanned Aerial Vehicles (UAVs) have an autopilot software that is a real-time system and requires rapid response to changing sensor data to keep the UAV under control. In this attack scenario, we consider ArduPilot [69], which is an open source autopilot software. Many UAV companies (e.g., 3DR, jDrones, PrecisionHawk, AgEagle, and Kespry) are deploying ArduPilot on their devices and large corporations (e.g., NASA, Intel, and Insitu/Boeing) use it for test and development.

ArduCopter (a version of ArduPilot customized to copters) has 19 main real-time tasks with deadline constraints. These tasks run under the context of a flight controller process (`main_loop`) and are scheduled us-

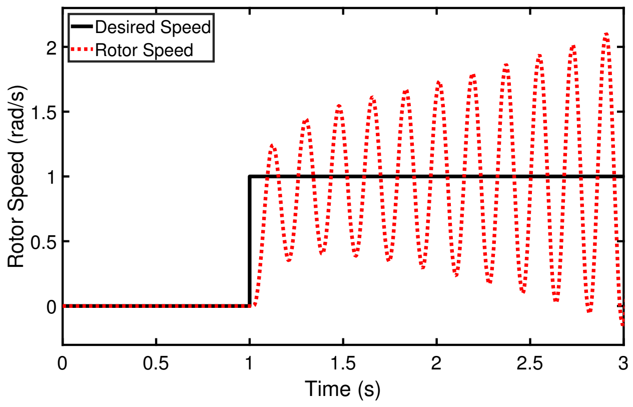


(a) DC motor rotational speed (stable).

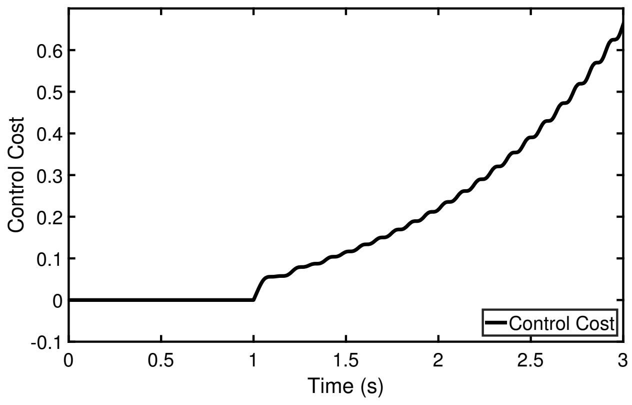


(b) DC motor control cost (stable).

Fig. 6: Stable DC motor without adversary.



(a) DC motor rotational speed (unstable).



(b) DC motor control cost (unstable).

Fig. 7: Unstable DC motor in the presence of adversary.

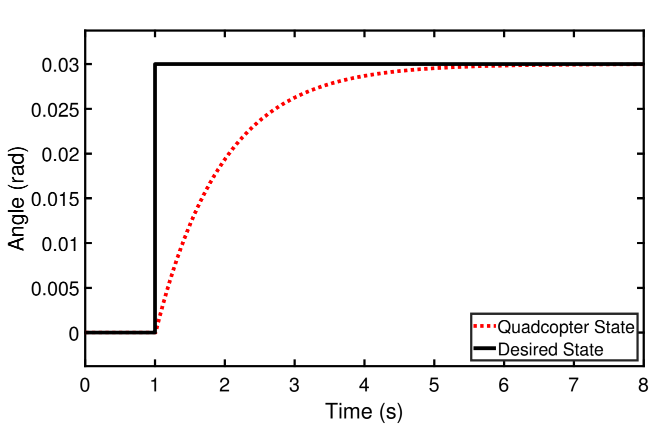
ing a real-time scheduling system. Each real-time task handles a specific flight control operation and executes a loop that has two real-time constraints: a period and a deadline. In this case-study, we consider two of the tasks running in the controller process, i.e., update_GPS and run_nav_updates. The update_GPS task reads the GPS signal from all GPS instances of the system. The run_nav_updates is a critical task that runs the autopilot task and actuates commands to the servos to control the UAV. Here, we consider the scenario where the adversary manipulates the timing of a less critical task like update_GPS, which will lead to an increase in jitter for the run_nav_updates task.

The update_GPS function (see Source Code 1) starts with a condition that if not satisfied, the main part of the code will not be executed. The condition checks if there is a new message received and if the status of the received signal satisfies previous assumptions. Therefore, if the attacker jams the GPS signal, then the entire update_GPS task will have a much smaller and negligible (close to 0) execution/computation time.

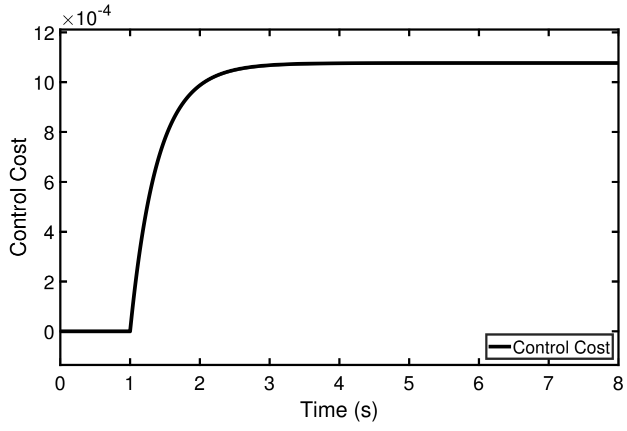
Source Code 1: update_GPS

```
static uint32_t last_gps_msg_ms;
static uint8_t ground_start_count = 5;
if (gps.last_message_time_ms() !=
    last_gps_msg_ms &&
    gps.status() >=
    AP_GPS::GPS_OK_FIX_3D) {
    last_gps_msg_ms =
    gps.last_message_time_ms();
    ...
```

Reducing the computation time of the update_GPS task can cause intolerable jitter for other tasks running on the same microprocessor (e.g., run_nav_updates). Note that in this scenario the attacker only decreases the computation time of update_GPS, which intuitively is harmless for other tasks and, therefore, is not prevented by the scheduler, but it will create large jitters for other tasks and may lead to instability of the UAV.

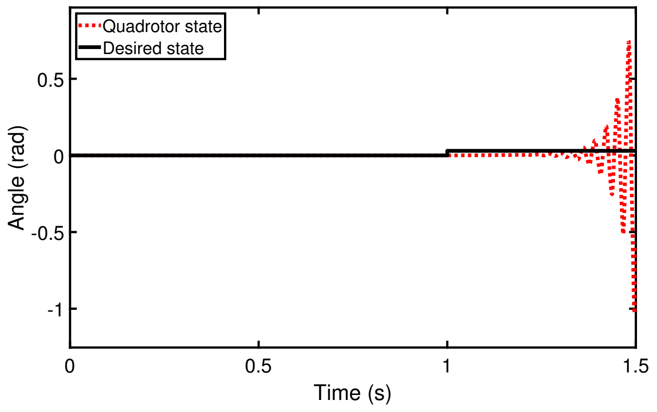


(a) Quadcopter vertical angle (stable).

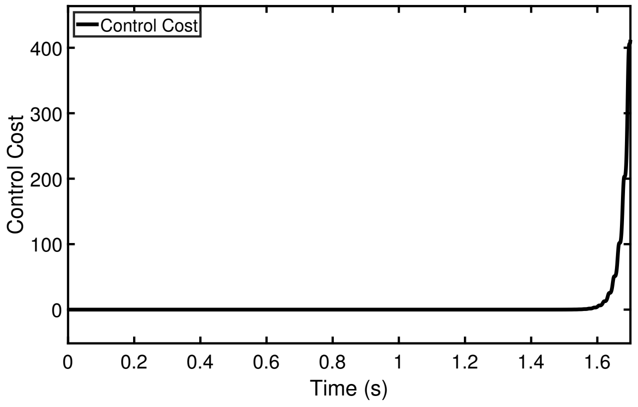


(b) Quadcopter control cost (stable).

Fig. 8: Stable quadcopter without adversary.



(a) Quadcopter vertical angle (unstable).



(b) Quadcopter control cost (unstable).

Fig. 9: Unstable quadcopter in the presence of adversary.

We consider a Butterfly attack scenario on a quadcopter running the flight control software and show experimentally how the attack would render the quadcopter unstable. We assume that the flight control software is running on the quadcopter's processor. Therefore, all the real-time tasks are scheduled on the shared processor. We focus only on the two `update_GPS` and `run_nav_updates` tasks running on the processor. The tasks have the following parameters,

$$\begin{aligned} \tau_1: \quad & \rho_1 = H, c_1 = 4, h_1 = 5, \\ \tau_2: \quad & \rho_2 = L, c_2 = 1, h_2 = 10, \end{aligned} \tag{13}$$

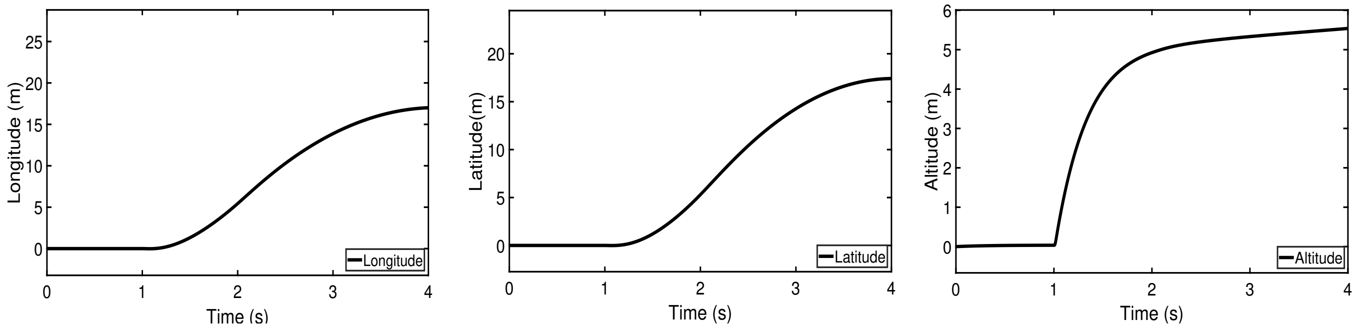
where task τ_2 is the task that is running the controller (`run_nav_updates`) to stabilize the quadcopter in the desired position and task τ_1 is the `update_GPS` task. Priorities are assigned according to RM scheduling.

We use the model provided in [70] to simulate the quadcopter in Simulink MATLAB. The model is linearized and can be written in state-space form as follows,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{14}$$

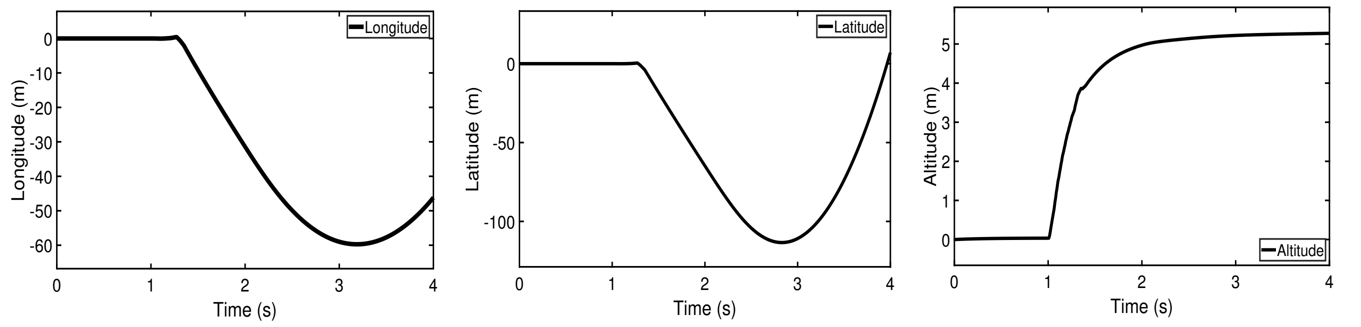
where $\boldsymbol{x}$ is the state of the quadcopter and shows its position and angle, and $\boldsymbol{u}$ is the input signal to the quadcopter. The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are calculated using the dynamic model of the quadcopter and depend on its physical properties, e.g., inertia moments in each axis, rotor inertia, propeller distance from center, and overall mass. The input signal is a vector of four elements (one for each rotor) and is calculated based on the current state of the quadcopter, such that the quadcopter is positioned in a desired coordinate and angle. The controller task receives the current and desired state of the quadcopter in each period and provides input signals to position the quadcopter in the desired coordinate.

If fixed-priority RM scheduling is applied, considering tasks described in Equation 13, the `run_nav_updates` task experiences a latency and jitter equal to 5ms and 0ms, respectively. With these values of latency and jitter,



(a) Longitude (b) Latitude (c) Altitude

Fig. 10: Position of quadcopter during transition.



(a) Longitude (b) Latitude (c) Altitude

Fig. 11: Position of quadcopter in the presence of adversary.

the quadcopter is stable. In our control–scheduling co-simulation framework, at time 1s we set the desired states (the horizontal angles) of the quadcopter. The controller manages to converge to the desired state after 6s (see Figure 8(a)) and the control cost is kept finite (see Figure 8(b)), which indicates the stability of the system.

To perform the Butterfly attack, we run the same simulation and change the desired state of the quadcopter at time 1s, in our control–scheduling co-simulation framework. However, this time, the attacker increases the period of the `update_GPS` (τ_1) from 5ms to 20ms. This is performed by jamming the GPS signal such that three out of four messages are not received. Note that, by increasing the period of τ_1, actually the available processor share for τ_2 is increased by reducing the interference it suffers. Intuitively, this should lead to better control performance of *run_nav_updates* (τ_2). Nevertheless, in the new schedule for τ_1 and τ_2 the latency of τ_2 decreases to 1ms but the jitter increases to 4ms. The quadcopter is not stable with this latency and jitter. After changing the desired state, the controller is unable to converge the state of the quadcopter into the desired state (see Figure 9(a)) and the control cost tends to infinity after 1s (see Figure 9(b)), which indicates the instability of the system.

The above example shows how Butterfly attack can destabilize a UAV. In the following, we show that, using this attack, the adversary is also able to hijack the UAV and move it to a different location. In the quadcopter model, the autopilot software changes the desired angles of the quadcopter to move it toward a desired location. We experimentally show that if the adversary interferes during this transition process and introduces a specific amount of jitter, then she is able to change the location of the quadcopter.

We again focus on two tasks running on the processor, as stated in Equation 13. Let us assume that the autopilot software changes the desired angle of the quadcopter at time 1s. As a result, the quadcopter will end up in position $(X = 17, Y = 17, Z = 5.5)$ at time 4s. Figure 10 shows the location of the quadcopter during this transition when there is no attack and the quadcopter reaches the desired location at time 4s. Figure 12 shows the path (blue line) the queadcopter follows to reach the destination.

In order to apply Butterfly attack, we simulate an adversary that periodically increases the period of the `update_GPS` task from 5 to 20 and decreases it back to 5 (e.g., by jamming 3 in every 4 GPS messages). This process is repeated for 50 times during the transition. Subsequently, the quadcopter does not become unstable, but it goes along an unexpected path (red line in Figure 12) and at some point during the attack it enters an unsafe zone, which is far from the point that the quadcopter is supposed to be,

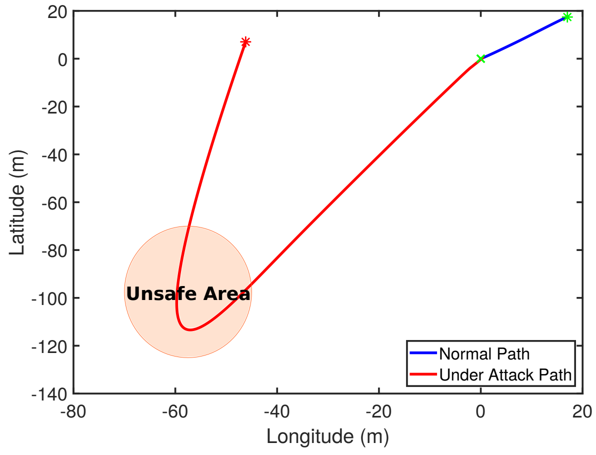


Fig. 12: Two-dimensional position of the quadcopter between time $0s$ and $4s$. The cross and star signs in green show the initial position $(0, 0)$ and desired destination $(17, 17)$ of the quadcopter, respectively. The blue line shows the path of the quadcopter when there is no attack and the red line shows its path when an attack is undergoing. The red star sign at the end of red path shows the position of the quadcopter after $4s$, when it is under attack.

where the quadcopter can be hijacked. After the attack, the quadcopter again starts to converge to the desired location that the autopilot software specified. Figure 11 shows the location of the quadcopter during the transition time. Note that the adversary has some control over the location of the quadcopter during this attack. Therefore, the adversary can direct the quadcopter into a desired zone by imposing certain amount of jitter. Figure 12 shows the two-dimensional position of the quadcopter over time, with and without the Butterfly attack.

VIII. Discussion

In this section, we discuss several possible mitigation, detection, and defense mechanisms against the Butterfly attack. To detect and mitigate the Butterfly attack, it is possible to synthesize a set of safety constraints during the design phase. This is similar to monitoring overruns in current operating systems, but captures more temporal aspects of the safety-critical applications, including the delay, latency, and jitter. Then, at runtime, these constraints will be monitored by an intrusion detection unit to identify potential threats and initiate the mitigation protocol.

The mitigation protocol may involve executing a dummy task to compensate for the timing manipulation introduced into the system by the adversary and, e.g., compensate for increased period or decreased execution time of a lower criticality task. In this approach, however, the resources are not used for running applications. One solution could be to promote low-priority tasks instead in order to avoid this phenomenon. Alternatively, the mitigation protocol may initiate a procedure to switch to more robust controllers, with of course worse control performance, for the safety-critical applications. This mitigation protocol requires careful planning of the switching procedure, due to the potentially pathological scenarios with switching control. In particular, it is well-known that switching between two stable controllers may still destabilize the control plant [71], if not done carefully.

Another mitigation technique is to assign priorities with respect to task criticality. By doing this the adversary is not able to interfere high-critical tasks by manipulating less-critical ones. Nonetheless, this apparently simple solution is very often not applicable: (1) In many cases the system designer needs to assign higher priority to less-critical tasks for schedulability reasons so that the less critical tasks can meet their deadlines. (2) This mitigation technique can only be applied to systems that use fixed-priority scheduling, while the Butterfly attack is applicable to other scheduling algorithms like EDF (Earliest Deadline First).

In the context of control design, the mitigation might be provided by designing a controller that is robust to temporal attacks [72], [73]. However, this might result in more complex and resource consuming controllers with potentially lower control quality. Essentially, this is because such controllers are designed for a scenario that does not correspond to the expected average-case scenario.

The ultimate solution to eliminate the possibility of such temporal attacks is to ensure temporal isolation using bandwidth allocation and control servers [20], [74]. The main drawback of control servers is either lack of any hard guarantees or over-provisioning resource allocation that may leave resources under-utilized, even as low as 50% of the total capacity. To address this issue, several online resource allocation mechanisms have been proposed in the literature [21]. Nevertheless, Miller and Valasek [58] demonstrate a remote attack against commercial automobiles [51], even when isolation, or so-called air-gap, between the entertainment and critical functionality in vehicles, is implemented and envisioned to reduce the attack possibilities. Therefore, a comprehensive study of mitigation and defense mechanisms for complex cyber-physical systems remains the topic of our future work.

IX. Related Work

The implementation of control applications on shared platforms consists of two main steps: controller synthesis and control task scheduling. The traditional design flow of such applications is based on the principle of *separation of concerns*. The main drawbacks of this principle are poor resource utilization and control performance [3]. This is due to the fact that the effects of the decisions made during one of the design steps are not considered on the decisions that are made during the second step, which often leads to suboptimal solutions.

The timing problems in real-time control systems were first brought up by Wittenmark et al. [2]. The authors discuss the issue of time-varying delays introduced during the implementation phase of such systems, which often lead to poor control performance and may even jeopardize the stability of control applications. Since then, several studies have considered implementation-induced timing issues in the design of such embedded control and cyber-physical systems [10]–[42].

Racu and Ernst [75] show that, contrary to what is widely believed, shorter execution time for one task of a task chain, mapped on a multicore platform, may lead to a longer end-to-end delay for the same task chain, in the context of purely hard real-time systems. Aminifar et al. [9] demonstrate that intuitively better design parameters, e.g., priority, period, and execution time, may lead to instability of control applications on shared platforms. Nevertheless, to date, the security exploitation of such anomalous scenarios has not been discussed in the context of embedded control and cyber-physical systems.

Due to the tight coupling in control systems between physical components and cyber unit many attacks have been reported on the control applications in cyber-physical systems [76]–[78]. These attacks are usually studied in two categories [79]–[81]: First, *deceptive* attacks, where the adversary interferes in the connection between sensor-controller-actuator and sends incorrect information [79], [82]–[85]. Second, *denial-of-service* attacks, where the adversary prevents sensor, controller, or actuator from receiving data [86], [87]. However, these studies do not target manipulation of the temporal properties of control applications.

Temporal attacks, on the other hand, which can affect control systems in many cases are not well studied in the literature. Temporal attacks, if not properly mitigated, can jeopardize the stability of the control systems by introducing non-ideal timing scenarios to the control application. The proposed temporal attacks in the literature can be classified into two categories. First, the attacks that target the communication of sensors and/or actuators with the controller and result in dropping messages that are being transferred [73], [88], [89]. Second, time delay attacks, that tamper the temporal characteristics of the communication channel in order to delay the signals received by certain nodes (e.g., sensor, actuator, or controller) [90]–[92]. However, in both categories to deliver the attack the adversary needs to change the timing behavior of the critical messages received by the critical application. A trivial detection mechanism can protect highly critical control applications from these attacks. This is very different from the Butterfly attack where the adversary interferes the lower criticality messages/tasks in order to affect the high criticality application.

X. Conclusion

In this paper, we introduce the Butterfly attack, a new attack scenario for cyber-physical systems. The Butterfly attack highlights the importance of the implementation aspects of cyber-physical systems on the underlying execution platforms and the temporal sensitivity of control applications implemented on shared platforms. Such resource sharing may lead to complex timing behaviors and, in turn, counter-intuitive timing anomalies that can then be exploited by an adversary to render a critical control system unstable, which may have major irreversible consequences. We demonstrate the possibility of such attacks in two case-studies from the automotive and avionic domains.

References

[1] M. D. Natale and A. L. Sangiovanni-Vincentelli, “Moving from federated to integrated architectures in automotive: The role of standards, methods and tools,” *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[2] B. Wittenmark, J. Nilsson, and M. Törngren, “Timing problems in real-time control systems,” in *Proceedings of the American Control Conference*, 1995, pp. 2000–2004.

[3] K. E. Årzén and A. Cervin, “Control and embedded computing: Survey of research directions,” in *Proceedings of the* 16th *IFAC World Congress*, 2005.

[4] A. Cervin, P. Pazzaglia, M. Barzegaran, and R. Mahfouzi, “Using jittertime to analyze transient performance in adaptive and reconfigurable control systems,” in *2019 IEEE 24th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019.

[5] E. A. Lee, “Cyber physical systems: Design challenges,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-8, 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html

[6] M. Wolf, A. Weimerskirch, and C. Paar, “Secure in-vehicle communication,” in *Embedded Security in Cars*, K. Lemke, C. Paar, and M. Wolf, Eds. Springer Berlin Heidelberg, 2006, pp. 95–109.

[7] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems*, 3rd ed. Prentice Hall, 1997.

[8] A. Cervin, “Stability and worst-case performance analysis of sampled-data control systems with input and output jitter,” in *Proceedings of the 2012 American Control Conference (ACC)*, 2012, pp. 1–10.

[9] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, “Stability-aware analysis and design of embedded control systems,” in *Proceedings of the* 11th *ACM International Conference on Embedded Software*, 2013, pp. 23:1–23:10.

[10] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, “On task schedulability in real-time control systems,” in *Proceedings of the* 17th *IEEE Real-Time Systems Symposium*, 1996, pp. 13–21.

[11] H. Rehbinder and M. Sanfridson, “Integration of off-line scheduling and optimal control,” in *Proceedings of the* 12th *Euromicro Conference on Real-Time Systems*, 2000, pp. 137–143.

[12] A. Cervin, B. Lincoln, J. Eker, K. E. Årzén, and G. Buttazzo, “The jitter margin and its application in the design of real-time control systems,” in *Proceedings of the* 10th *International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004, pp. 1–10.

[13] T. Nghiem, G. J. Pappas, R. Alur, and A. Girard, “Time-triggered implementations of dynamic controllers,” in *Proceedings of the* 6th *ACM & IEEE International conference on Embedded software*, 2006, pp. 2–11.

[14] E. Bini and A. Cervin, “Delay-aware period assignment in control systems,” in *Proceedings of the* 29th *IEEE Real-Time Systems Symposium*, 2008, pp. 291–300.

[15] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, “Task scheduling for control oriented requirements for cyber-physical systems,” in *Proceedings of the* 29th *IEEE Real-Time Systems Symposium*, 2008, pp. 47–56.

[16] P. Naghshtabrizi and J. P. Hespanha, “Analysis of distributed control systems with shared communication and computation resources,” in *Proceedings of the 2009 American Control Conferance (ACC)*, 2009.

[17] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele, “A hybrid approach to cyber-physical systems verification,” in *Proceedings of the* 49th *Design Automation Conference*, 2012.

[18] G. Mancuso, E. Bini, and G. Pannocchia, "Optimal priority assignment to control tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, p. 161, 2014.

[19] T. Bund and F. Slomka, "A delay density model for networked control systems," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, 2013, pp. 205–212.

[20] A. Aminifar, E. Bini, P. Eles, and Z. Peng, "Analysis and desing of real-time servers for control applications," *IEEE Transactions on Computers*, 2015.

[21] A. Aminifar, P. Eles, and Z. Peng, "Jfair: A scheduling algorithm to stabilize control applications," in *Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015.

[22] Y. Xu, K. E. Årzén, A. Cervin, E. Bini, and B. Tanasa, "Exploiting job response-time information in the co-design of real-time control systems," in *IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015, pp. 247–256.

[23] K. E. Årzén, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000, pp. 4865–4870.

[24] M. M. Ben Gaid, A. Cela, and Y. Hamam, "Optimal integrated control and scheduling of networked control systems with communication constraints: Application to a car suspension system," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 4, pp. 776–787, 2006.

[25] L. Palopoli, C. Pinello, A. Bicchi, and A. Sangiovanni-Vincentelli, "Maximizing the stability radius of a set of systems under real-time scheduling constraints," *Automatic Control, IEEE Transactions on*, vol. 50, no. 11, pp. 1790–1795, 2005.

[26] M. Behnam and D. Isovic, "Real-time control and scheduling co-design for efficient jitter handling," in *Proceedings of the 13th IEEE International Conference onEmbedded and Real-Time Computing Systems and Applications*, 2007, pp. 516–524.

[27] M. Lemmon, T. Chantem, X. S. Hu, and M. Zyskowski, "On self-triggered full-information h-infinity controllers," in *Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control*, 2007, pp. 371–384.

[28] S. Samii, A. Cervin, P. Eles, and Z. Peng, "Integrated scheduling and synthesis of control applications on distributed embedded systems," in *Proceedings of the Design, Automation and Test in Europe Conference*, 2009, pp. 57–62.

[29] G. Mancuso, E. Bini, and G. Pannocchia, "Optimal computational resource allocation for control task under fixed priority scheduling," *Proceedings of the 18th IFAC World Congress*, vol. 18, 2011.

[30] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-triggered implementations of mixed-criticality automotive software," in *Proceedings of the 15th Conference for Design, Automation and Test in Europe (DATE)*, 2012.

[31] D. Fontantelli, L. Palopoli, and L. Greco, "Optimal cpu allocation to a set of control tasks with soft real–time execution constraints," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, 2013, pp. 233–242.

[32] D. Fontanelli, L. Greco, and L. Palopoli, "Soft real-time scheduling for embedded control systems," *Automatica*, vol. 49, no. 8, pp. 2330 – 2338, 2013.

[33] D. Fontanelli, L. Palopoli, and L. Abeni, "The continuous stream model of computation for real–time control," in *Proceedings of the 34th IEEE Real-Time Systems Symposium, Vancouver, Canada*, 2013.

[34] A. Aminifar, P. Eles, Z. Peng, and A. Cervin, "Control-quality driven design of cyber-physical systems with robustness guarantees," in *Proceedings of the 16th Conference for Design, Automation and Test in Europe (DATE)*, 2013.

[35] Y. Xu, K. E. Årzén, E. Bini, and A. Cervin, "Response time driven design of control systems," in *19th IFAC World Congress*, Cape Town, South Africa, Aug. 2014.

[36] N. Dhruva, P. Kumar, G. Giannopoulou, and L. Thiele, "Computing a language-based guarantee for timing properties of cyber-physical systems," in *Proceedings of the 17th Conference for Design, Automation and Test in Europe (DATE)*, 2014.

[37] A. Biondi and G. Buttazzo, "Modeling and analysis of engine control tasks under dynamic priority scheduling," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4407–4416, 2018.

[38] A. Biondi, M. Di Natale, and G. Buttazzo, "Response-time analysis of engine control applications under fixed-priority scheduling," *IEEE Transactions on Computers*, vol. 67, no. 5, pp. 687–703, 2017.

[39] W. Chang, D. Goswami, S. Chakraborty, and A. Hamann, "Os-aware automotive controller design using non-uniform sampling," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 4, p. 26, 2018.

[40] D. Ziegenbein and A. Hamann, "Timing-aware control software design for automotive systems," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 56.

[41] A. Aminifar, P. Eles, Z. Peng, A. Cervin, and K.-E. Årzén, "Control-quality driven design of embedded control systems with stability guarantees," *IEEE Design and Test*, 2017.

[42] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén, "Optimal harmonic period assignment: complexity results and approximation algorithms," *Real-Time Systems*, vol. 54, no. 4, pp. 830–860, 2018.

[43] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 239–243.

[44] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2018.

[45] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002, pp. 1319–1324.

[46] J. Palencia Gutierrez, J. Gutierrez Garcia, and M. Gonzalez Harbour, "Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems," in *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, 1998, pp. 35–44.

[47] O. Redell and M. Sanfridson, "Exact best-case response time analysis of fixed priority scheduled tasks," in *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, 2002, pp. 165–172.

[48] D. Davidson, H. Wu, R. Jellinek, V. Singh, and T. Ristenpart, "Controlling uavs with sensor input spoofing attacks," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[49] E. Deligne, "Ardrone corruption," *Journal in Computer Virology*, vol. 8, no. 1-2, pp. 15–27, 2012.

[50] D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Drone hack: Spoofing attack demonstration on a civilian unmanned aerial vehicle," 2012.

[51] A. Drozhzhin, "Black hat usa 2015: The full story of how that jeep was hacked." [Online]. Available: https://www.kaspersky.com/blog/blackhat-jeep-cherokee-hack-explained/9493/

[52] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[53] K. Highnam, K. Angstadt, K. Leach, W. Weimer, A. Paulos, and P. Hurley, "An uncrewed aerial vehicle attack scenario and trustworthy repair architecture," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 222–225.

[54] Y. Son, H. Shin, D. Kim, Y. Park, J. Noh, K. Choi, J. Choi, and Y. Kim, "Rocking drones with intentional sound noise on gyroscopic sensors," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 881–896.

[55] C. Ferdinand and R. Heckmann, *aiT: Worst-Case Execution Time Prediction by Static Program Analysis*. Springer US, 2004, pp. 377–383.

[56] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem–overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 36, 2008.

[57] 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 *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[58] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[59] J. Eltze, "Double-can controller as bridge for different can networks," in *Proceedings of the 4th International CAN Conference*, 1997.

[60] K. Karray, J.-L. Danger, S. Guilley, and M. A. Elaabid, “Identifier randomization: An efficient protection against can-bus attacks,” in *Cyber-Physical Systems Security*. Springer, 2018, pp. 219–254.

[61] A. Taylor, N. Japkowicz, and S. Leblanc, “Frequency-based anomaly detection for the automotive can bus,” in *2015 World Congress on Industrial Control Systems Security (WCICSS)*. IEEE, 2015, pp. 45–49.

[62] M. Marchetti and D. Stabili, “Anomaly detection of can bus messages through analysis of id sequences,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1577–1583.

[63] D. Stabili, M. Marchetti, and M. Colajanni, “Detecting attacks to internal vehicle networks through hamming distance,” in *2017 AEIT International Annual Conference*. IEEE, 2017, pp. 1–6.

[64] R. Mahfouzi, A. Aminifar, P. Eles, Z. Peng, and M. Villani, “Intrusion-damage assessment and mitigation in cyber-physical systems for control applications,” in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM, 2016, pp. 141–150.

[65] G. Weiss, P. Schleiss, and C. Drabek, “Towards flexible and dependable e/e-architectures for future vehicles,” in *4th International Workshop on Critical Automotive Applications: Robustness & Safety (CARS 2016)*, 2016.

[66] M. Maul, G. Becker, and U. Bernhard, “Service-oriented ee zone architecture key elements for new market segments,” *ATZelektronik worldwide*, vol. 13, no. 1, pp. 36–41, 2018.

[67] B. Messner and D. Tilbury, “Control tutorials for matlab and simulink - motor speed: System modeling,” http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling, (Accessed on 02/15/2019).

[68] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. E. Årzén, “How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime,” *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, 2003.

[69] “Ardupilot open source,” http://ardupilot.org/, accessed: 2018-12-07.

[70] S. Bouabdallah, “Design and control of quadrotors with application to autonomous flying,” p. 155, 2007. [Online]. Available: http://infoscience.epfl.ch/record/95939

[71] J. P. Hespanha and A. S. Morse, “Switching between stabilizing controllers,” *Automatica*, vol. 38, no. 11, pp. 1905–1917, 2002.

[72] M. Pajic, J. Weimer, N. Bezzo, P. Tabuada, O. Sokolsky, I. Lee, and G. J. Pappas, “Robustness of attack-resilient state estimators,” in *ICCPS’14: ACM/IEEE 5th International Conference on Cyber-Physical Systems (with CPS Week 2014)*. IEEE Computer Society, 2014, pp. 163–174.

[73] Y. Shoukry, J. Araujo, P. Tabuada, M. Srivastava, and K. H. Johansson, “Minimax control for cyber-physical systems under network packet scheduling attacks,” in *Proceedings of the 2nd ACM international conference on High confidence networked systems*. ACM, 2013, pp. 93–100.

[74] A. Cervin and J. Eker, “The control server: A computational model for real-time control tasks,” in *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.* IEEE, 2003, pp. 113–120.

[75] R. Racu and R. Ernst, “Scheduling anomaly detection and optimization for distributed systems with preemptive task-sets,” in *Proceedings of the* 12^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006, pp. 325–334.

[76] J. Slay and M. Miller, “Lessons learned from the maroochy water breach,” in *International Conference on Critical Infrastructure Protection*. Springer, 2007, pp. 73–82.

[77] J. P. Farwell and R. Rohozinski, “Stuxnet and the future of cyber war,” *Survival*, vol. 53, no. 1, pp. 23–40, 2011.

[78] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, “Attacks against process control systems: risk assessment, detection, and response,” in *Proceedings of the 6th ACM symposium on information, computer and communications security*. ACM, 2011, pp. 355–366.

[79] C. Kwon, W. Liu, and I. Hwang, “Security analysis for cyber-physical systems against stealthy deception attacks,” in *2013 American control conference*. IEEE, 2013, pp. 3344–3349.

[80] A. A. Cardenas, S. Amin, and S. Sastry, “Secure control: Towards survivable cyber-physical systems,” in *2008 The 28th International Conference on Distributed Computing Systems Workshops*. IEEE, 2008, pp. 495–500.

[81] Y. Z. Lun, A. DâĂŹInnocenzo, F. Smarra, I. Malavolta, and M. D. Di Benedetto, “State of the art of cyber-physical systems security: An automatic control perspective,” *Journal of Systems and Software*, vol. 149, pp. 174–216, 2019.

[82] S. Amin, X. Litrico, S. S. Sastry, and A. M. Bayen, “Stealthy deception attacks on water scada systems,” in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. ACM, 2010, pp. 161–170.

[83] Y. Liu, P. Ning, and M. K. Reiter, “False data injection attacks against state estimation in electric power grids,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 13, 2011.

[84] F. Miao, Q. Zhu, M. Pajic, and G. J. Pappas, “Coding schemes for securing cyber-physical systems against stealthy data injection attacks,” *IEEE Transactions on Control of Network Systems*, vol. 4, no. 1, pp. 106–117, 2016.

[85] I. Jovanov and M. Pajic, “Relaxing integrity requirements for attack-resilient cyber-physical systems,” *IEEE Transactions on Automatic Control*, 2019.

[86] S. Amin, A. A. Cárdenas, and S. S. Sastry, “Safe and secure networked control systems under denial-of-service attacks,” in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2009, pp. 31–45.

[87] A. Gupta, C. Langbort, and T. Başar, “Optimal control in the presence of an intelligent jammer with limited actions,” in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 1096–1101.

[88] G. Fiore, Y. H. Chang, Q. Hu, M. D. Di Benedetto, and C. J. Tomlin, “Secure state estimation for cyber physical systems with sparse malicious packet drops,” in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 1898–1903.

[89] J. Moon and T. Başar, “Minimax control over unreliable communication channels,” *Automatica*, vol. 59, pp. 182–193, 2015.

[90] G. Bianchin and F. Pasqualetti, “Time-delay attacks in network systems,” in *Cyber-Physical Systems Security*. Springer, 2018, pp. 157–174.

[91] X. Lou, C. Tran, R. Tan, D. K. Yau, and Z. T. Kalbarczyk, “Assessing and mitigating impact of time delay attack: a case study for power grid frequency control,” in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*. ACM, 2019, pp. 207–216.

[92] K. Rahimi, A. Parchure, V. Centeno, and R. Broadwater, “Effect of communication time-delay attacks on the performance of automatic generation control,” in *2015 North American Power Symposium (NAPS)*. IEEE, 2015, pp. 1–6.