Statistical model checking for the analysis of attacks in connected autonomous vehicles

***Abstract*—This paper proposes an approach for the analysis of the effects of attacks in connected autonomous vehicles by simulated attack injection and statistical model checking technique. The dynamic vehicles, together with the co-ordination algorithm among vehicles and the attacks are formalized using hybrid automata in UPPAAL framework. Then, the statistical model checker, UPPAAL-SMC, allows to study the resilience of the vehicle's system to attacks across a range of circumstances and uncertainties. The approach is applied to a platoon of vehicles, and properties of the system under attack in case of various driver patterns of the platoon's leader and parameters of a data alteration attack are analyzed.**

***Index Terms*—Statistical Model Checking, Autonomous Vehicles, Cybersecurity, UPPAAL, Platoon Resilience**

I. Introduction

Cybersecurity is a really important topic for connected autonomous vehicles, since they are highly computerized and connected to the network, thus providing a wide range of access points for a potential attacker, who could gain full control over the vehicle and turn off all safety measures installed on it [1]. Many attack types have been developed and tested on a vehicle, proving that the considered vulnerabilities could actually be exploited by a potential attacker.

Platooning [2] is a driving strategy where multiple vehicles travel closely together in a coordinated group, enhancing road safety, lowering emissions, and reducing driver workload. The main goal of a platoon system is to maintain a predefined inter-vehicle distance determined by a spacing policy, and to guarantee the string stability, i.e. to avoid perturbation propagating along the vehicles' string. These two objectives are obtained through vehicle cooperation. Traditionally, platoon vehicles cooperation is performed using vehicle to vehicle (V2V) communication, exploiting IEEE 802.11p standard. The advent of 5G has enabled centralized approaches that combine cellular communications and multi-access edge computing (MEC), as proposed in [3], [4].

UPPAAL [5] is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of *timed automata* [6]. A timed automaton consists of a set of states (locations) and transitions, and a set of real-valued variables for measuring time between transitions, named Clocks. Invariants can be added to locations to specify timing constraints on leaving the locations.

A vehicle is a cyber-physical system (CPS), where digital and physical components interact. Connected vehicles are complex CPSs which communicate through a network and they are generally characterized by unpredictability and variability.

From the point of view of modelling, in UPPAAL physical quantities and their derivatives can be modelled with clocks; and for what concerns the analysis of system's properties, Statistical Model Checking (SMC) can be used [7]. SMC encompasses a range of techniques that monitor several executions of the system with respect to specific property, leveraging statistical methods to estimate the overall correctness of the model. In particular, SMC estimates whether a given property holds with a specified level of confidence using results from the statistics area. For example, [8] applied this approach to power electrical appliances; in [9] SMC is used to validate the safety properties of an autonomous vehicle controller for switching lane on a motorway; in [10] this approach has been used to model the stochastic nature of cooperative systems, in joining and leaving a vehicle platoon.

In this work, we present a novel approach to analyse security of connected autonomous vehicles based on (i) model-based attack injection in the system using the UPPAAL formal modeling framework and (ii) exploration of the effects of attacks to the system using the UPPAAL SMC. We show the application to a platoon of vehicles, in case of data alteration attack to state parameters of the vehicles. The resilience of the platoon to an attack is analyzed, stochastically varying the driving and the attack parameters. The approach is general and can be used to analyse the resilience of the system to an anticipated set of attacks.

The paper is structured as follows: Section II illustrates basic concepts used in this work, Section III shows the network of timed automata modeling the platoon and attacks, Section IV uses UPPAAL to simulate the behavior of the platoon, Section V uses UPPAAL SMC's probabilistic queries to analyze properties of the platoon and Section VI includes a discussion and conclusions.

II. BACKGROUND

This section briefly introduces the platooning application and the formal modelling framework UPPAAL through a simple example.

A. Platooning

We consider a generic platoon remotely managed by an edge platoon controller (MEC approach), as described in [3] and reported in Fig. 1. There are four vehicles moving along one axis forming a platoon, one leader's driver and three follower vehicles. Each car is indexed from 0 to 3, starting from the leader with 0. In our model, we use the Cooperative

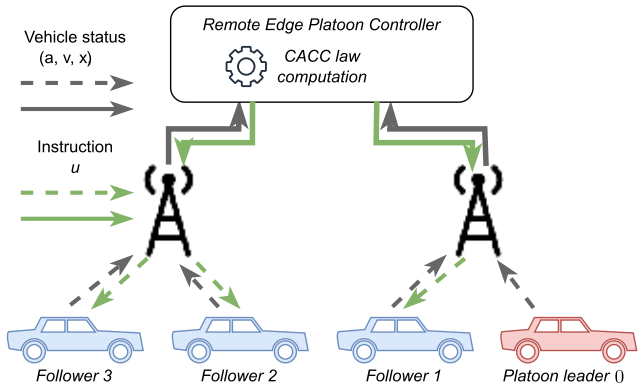


Figure 1: Edge-based platoon control schema

Adaptive Cruise Control (*CACC*) [11] for the co-ordination of the platoon.

Each vehicle sends its kinematic status, e.g. velocity, to the remote controller hosted in an edge server through the mobile network. The the controller collects the status of all cars, after which the CACC control law is applied to compute the desired acceleration of each follower. The desired accelerations are then sent back to the cars.

In particular, each vehicle i is characterized by four scalar values $(u, a, v, x)_i$. The first value u_i is the car's input value and represents the vehicle's desired acceleration: for the leader (value u_0), it is a known function part of the problem's hypothesis, whereas for the other cars it's the output of a control system.

The other values are the output current values and represent for each vehicle, respectively, the current acceleration a_i, the velocity v_i and the position x_i. These scalars can be conceptualized as *vehicle sensors' readings*, a term that will be used throughout the rest of the paper.

CACC operates under a constant-spacing policy and guarantees the string stability. Let D be the desired distance between car i and the car in front $i-1$. Formally, the instruction for the i-th vehicle, with $i > 0$, is given by the formula

$$\begin{aligned} u_i = &\alpha_1 a_{i-1} + \alpha_2 a_0 + \\ &\alpha_3 (v_i - v_{i-1}) + \alpha_4 (v_i - v_0) + \\ &\alpha_5 (D - d_i), \end{aligned} \tag{1}$$

where d_i is the distance to the vehicle in front ($d_i = x_{i-1} - x_i - l$), with l the length of the car), and α_i are control gain parameters. We assume in our case-study, $\alpha = (0.5, 0.5, -0.3, -0.1, -0.04)$.

The desired acceleration (also named reference acceleration) for vehicle i, depends on the acceleration of the vehicle in front; the acceleration of the leader; the difference of velocity between the vehicle and the vehicle in front; the difference of velocity between the vehicle and the leader; the gap between the desired distance D and the distance to the vehicle in front. The overall behavior of the platoon is imposed by u_0, the acceleration command given to the leader, which is a signal dependent only on time.

B. Uppaal SMC

Consider the simple system, composed by two cars, one following the other. The leader car (Driver) moves according to a given driving scenario and imposes the acceleration on the follower car (Car).

Each car is represented by an automaton. The acceleration imposed on Car is defined by a global variable `a`, set by the Driver. The clock variable `t`, defines interleaving time between transitions.

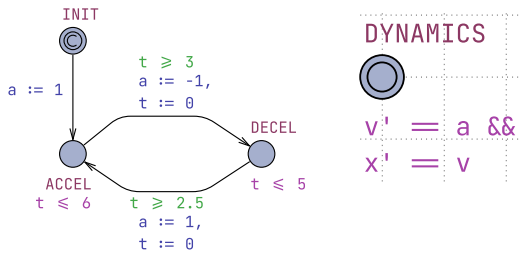
Fig. 2 (a) models the Driver. `INIT` is the initial location. The driver's behavior is modeled as follows. The first transition is enabled and set the acceleration to $a = 1\,\frac{\text{m}}{\text{s}^2}$. The location `INIT` is marked with a **C** that stands for *committed*, time is not allowed to pass when a process is in a committed location. In the example, it forces the automaton to transition to state `ACCEL` and thus assigns `a := 1` immediately. Then, invariant $t \leq 6$ assigned to location `ACCEL` and guard $t \geq 3$ of the transition exiting the location guarantees that the transition is executed in the time range $[3, 6]$. The execution of the transition assigns a new value to the global variable `a` and the clock `t` is reset to 0. A similar behaviour is exhibited when the system is in location `DECEL`, except that the state invariant is $t \leq 5$ and the guard is $t \geq 2.5$. The Driver alternates an acceleration period of length $T_A \in [3, 6]$ with $a = 1\,\frac{\text{m}}{\text{s}^2}$ and a deceleration period of length $T_D \in [2.5, 5]$ with $a = -1\,\frac{\text{m}}{\text{s}^2}$.

Fig. 2 (b), models Car. It consists in a single location `DYNAMICS`. The follower car's speed and position are modeled by two clocks `x` and `v`, which are used to model the physics via the Lagrangian derivative notation. In particular, the state has an invariant that consists of the conjunction of the following two formulae: the derivative of the velocity is always equal to the acceleration; the derivative of the position is always equal to the velocity. During the execution of the system, all clocks are updated accordingly, in particular the car's automaton updates the velocity and position of the car.

Suppose we want to compute the probability of the velocity will exceed a given threshold within a specified time range: $\exists\ v$ in the period $[55, 60]$ seconds such that $v > 10\,\frac{\text{m}}{\text{s}}$. A possible UPPAAL SMC query formulation is

`Pr[t_sim <= 60] (<> t_sim >= 55 && car.v > 10)`,

where `Pr[t_sim <= 60]` selects simulation traces in the first 60 s, and `(<> t_sim >= 55 && car.v > 10)` selects paths



(a) Driver (b) Car

Figure 2: A simple UPPAAL system

Table I: Global clocks and variables

Name	Type	Description
t_sim	clock	Time t elapsed
a[N_CARS]	clock[]	Sensors' accelerations
v[N_CARS]	clock[]	Sensors' speeds
x[N_CARS]	clock[]	Sensors' positions
a_p[N_CARS]	clock[]	Cars' real accelerations
v_p[N_CARS]	clock[]	Cars' real velocity
x_p[N_CARS]	clock[]	Cars' real positions
a_n[N_CARS]	double[]	Readings after net. delays
v_n[N_CARS]	double[]	Readings after net. delays
x_n[N_CARS]	double[]	Readings after net. delays
u[N_CARS]	double[]	Desired accel.
u_n[N_CARS]	double[]	Desired accel. after net. delay
cacc	broadcast chan	CACC synchronization
crashed[N_CARS]	bool[]	Crashed cars
A	double[]	Attack amplitude

in which there exists a state satisfying the condition on simulation time >= 55 and car velocity > 10.

Such a query yields an estimated probability of $74\,\%$ with confidence interval of $95\,\%$. The desired confidence interval is a parameter of the program.

We note also that the rate λ of an exponential distribution can be assigned to a location. Then the probability of leaving the location is distributed according to the exponential distribution $P(\text{leaving after } t) = 1 - \exp(-\lambda t)$, that is the location is left on average after λ^{-1} time. This will be used in our work. For more information on UPPAAL, the reader can refer to [12].

III. MODELING

The platoon system's main objective is to keep all cars equidistant from each other, according to a safety distance, while keeping the same pace as the leader.

During a run, all follower cars' values for the desired accelerations u_i are computed simultaneously by the time-discrete CACC Controller. These values are then transmitted to the respective cars with a certain delay; consequently the cars' time evolutions are altered by these new values. The new values of cars' sensors' readings are transmitted back to the CACC, with a certain delay.

In this section, we will describe in detail how the input and output quantities to the CACC are defined and how the alteration attacks of the vehicle sensor's readings are built on them; then we will describe the network of timed automata for the platoon in Fig. 1.

A. Global clocks and variables

A summary of global clocks and variables is shown in Table I.

The system has one global clock t_sim that represents the time t elapsed from the beginning of a simulation, it is never stopped or changed.

The physical quantities representing the both sensors' readings and the actual quantities of the various cars composing the platoon, are modeled as UPPAAL's clocks, six of them for each car, namely, for the i-th car: a[i], v[i], x[i], a_p[i], v_p[i] and x_p[i]; where the _p variants represent the physical values read without any alteration, these are the values noted in $\hat{a}, \hat{v}, \hat{x}$ that we will introduce later in the text.

The input values u_i are modeled as an array of doubles u[N_CARS], u_n[N_CARS], with N_CARS the number of cars. Such an approach was taken because CACC is a time-discrete controller, that is it updates its outputs at discrete intervals rather than continuously.

The *broadcast channel* cacc is used as an action to synchronize the various CACC units.

The i-th elements of the boolean vector crashed[N_CARS] are set to true after the i-th car has either rear-ended or has been rear-ended. The global variable A represents the attack amplitude A.

B. Modelling attacks

Regarding attacks, we consider attacks that adds spurious signals on top of the car sensors' readings, i.e. data sent by the car to the controller at the edge, through the mobile network. Attacks that affects the data sent back by the controller to cars can be modeled in a similar way.

Firstly, we have to introduce three new values associated to each car, $(\hat{a}, \hat{v}, \hat{x})_i$, these are the car's unaltered physical values and are necessary to model the physical evolution of the cars. Then, we have to specify the attack start time and the effect of the attack on the data sent to the controller.

Assume t_A to be the time at which the attack starts. Let us consider the case of a low-frequency sinusoidal attack of $1\,\text{Hz}$ of frequency and of amplitude A, added on top of the true car's acceleration signal $\hat{a}$ as shown in (2) — the coefficient $\frac{2\pi}{10}$ is the frequency expressed in radians per second. Additionally, a small bias $\frac{5}{\pi}A$ is introduced in (3).

The sensors' readings are altered, when $t \geq t_A$, as shown in equations (2), (3), (4).

$$a(t) = \hat{a}(t) + A\sin\left(\tfrac{2\pi}{10}t\right) \tag{2}$$

$$v(t) = \hat{v}(t) - \tfrac{5}{\pi}A\cos\left(\tfrac{2\pi}{10}t\right) + \tfrac{5}{\pi}A \tag{3}$$

$$x(t) = \hat{x}(t) - \tfrac{25}{\pi^2}A\sin\left(\tfrac{2\pi}{10}t\right) + \tfrac{5}{\pi}At. \tag{4}$$

The sensors' values are modified coherently, assuring that the relation $x' = v, v' = a$ holds true. The problem is made interesting by the signal's construction, which is designed to be difficult for an external observer to detect.

If car 1 is sending spurious data, then the control system of car 2 sees, for example, an erroneous distance $d_2(t) = x_2(t) - x_1(t) - l$.

C. Network of timed automata

1) Simulation rate: The Simulation automaton, shown in Fig. 3, is used to update, every $T = 0.01\text{s}$, the simulation's global variables that track collisions between vehicles and to synchronize the CACC's computation; the former is done by calling the *test_collision()* function and the latter by emitting a cacc! broadcast synchronization.

Since we are considering a centralized controller located in the edge, the various instances of the CACC law have to work

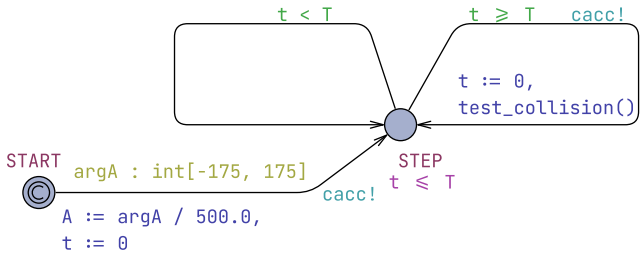


Figure 3: The Simulation automaton

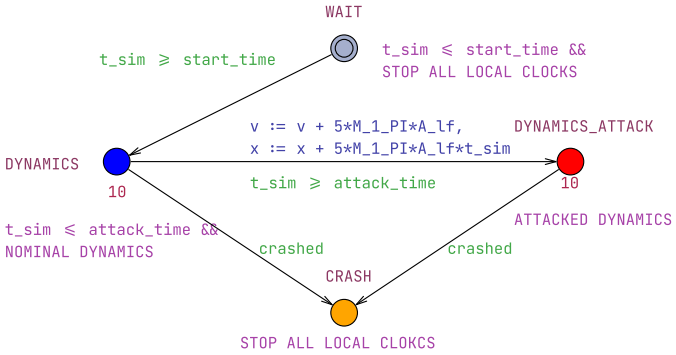


Figure 4: The template CarDynamics models the car's dynamics

as a single unit: they synchronize on cacc? — as described in Section III-C4.

Concerning the attacks, the Simulation automaton sets a value to A. The committed initial location START is connected to location STEP via a transition that sets the value of A by drawing it from a discrete uniform distribution.

The Simulation's STEP location has a looping edge with guard t >= T, that ensures the synchronization with the controller is executed exactly every T seconds.

2) Car's dynamics: The template CarDynamics, as shown in Fig. 4, models the car's dynamics. Each template instance models a car, one for the leader and three for the followers. There are four states:

- WAIT represents the car waiting for its time to start moving after start_time seconds (STOP ALL LOCAL CLOCKS is the state invariant)
- DYNAMICS models the time evolution of the vehicle given the input signal u (NOMINAL DYNAMICS is the state invariant)
- DYNAMICS_ATTACK is the same as the former but with attacks on the sensors' readings, as described above (ATTACK DYNAMICS is the state invariant)
- CRASH represents a crash with another vehicle (STOP ALL LOCAL CLOCKS is the state invariant)

The attacks were implemented by adding a new location, named DYNAMICS_ATTACK, in which the sensors' readings are modified accordingly. If $t_A = +\infty$ then no attack takes place as the guard t_sim >= attack_time of the transition from location DYNAMICS to DYNAMICS_ATTACK is always false. The

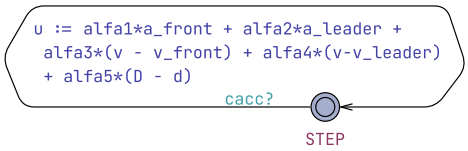


Figure 5: The CACC automaton implements the CACC

transition updates clocks x and v to set the initial conditions required by (3) and (4) with v := v + 5*M_1_PI*A and x := x + 5*M_1_PI*A*t_sim; the former adds $\frac{5}{\pi}$ and the latter adds $\frac{5}{\pi}t_A$ — the symbol M_1_PI is the floating-point representation of $\frac{1}{\pi}$.

The state invariants are not reported in the figure for brevity.

When a collision involving the car is detected, the guard crashed is satisfied, eventually causing a transition to the state CRASH; the locations modeling the dynamics of the vehicle have the exponential rate set to 10. The local variable crashed of car i is a reference to the global variable crashed[i].

3) Driving scenario: The automaton Driver models the driving pattern of the leader. This automaton generates the u_0 acceleration signal fed to the leader car. Three different operational modes, chosen stochastically with equiprobability $\frac{1}{3}$. The operational modes are:

1) the driver drives up to speed with constant acceleration
2) the driver drives up to speed with linear acceleration
3) the driver alternates periods of constant acceleration, no acceleration and constant deceleration

The parameters of said accelerations are set randomly at the start. For brevity's sake, we omitted this automaton. An example of simple Driver automaton is shown in section II, Fig. 2(a).

4) Adaptive Cruise Control: The controller at the edge is modeled by the CACC template: this automaton implements the CACC laws, for a vehicle. As shown in Fig. 5, all CACC instances are tied together by a cacc? broadcast synchronization channel. The transition is executed upon synchronization on the cacc! action from the Simulation automaton.

5) Communication network: The communication network is modeled as follows:

- the UPLINK template: this automaton models the transmission delay when sending a message from car to CACC
- the DOWNLINK template: this automaton models the transmission delay when sending a message from CACC to car

The transmission delay is modeled using an exponential rate of λ. We used a value to model an average delay of $\lambda^{-1} = 33\,\mathrm{ms}$ for all instances of both templates. For brevity's sake, we omitted the timed automaton UPLINK and DOWNLINK.

6) Conditions on the clocks: It is possible to model the dynamics by using the UPPAAL SMC's Lagrangian derivative notation on clocks to model a system of differential equations. In particular, in the DYNAMICS' invariant, — in logical conjunction to other things — we impose x' == v && v' == a and the acceleration prime derivative as a' == k*o + (u - o)/0.2 - k*a. An auxiliary clock o is defined in the

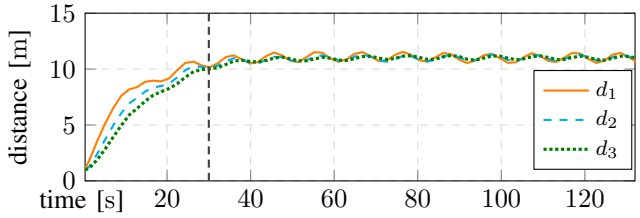


Figure 6: Distances between cars as time passes (no attack)

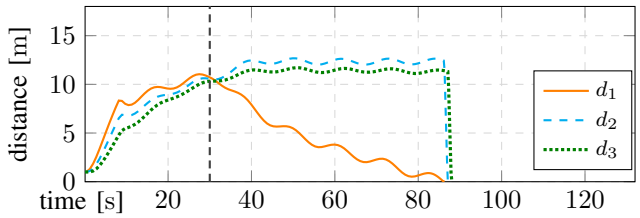


Figure 7: Distances between cars as time passes with $A < 0$

template with the condition in the invariant as `o' == (u - o)/0.2`, that is a first-order control system that models the actuation delay of the car. The clocks conditions in the location `DYNAMICS_ATTACK` are altered accordingly to (2), (3), (4).

IV. UPPAAL Simulation

In this section we use UPPAAL's query `simulate` to simulate the behavior of the platoon.

We show three possible behaviors of the platoon, plotting the distance d_i kept between pairs of consecutive cars. We consider a base case in which no attack takes place (that is $t_A = +\infty$ for all cars), and two cases with attack at $t_A = 30\,\mathrm{s}$ on car 1 with $A = \pm 0.08$.

The results of the three simulations are shown in Fig. 6, 7, 8.

In the first 20 to 25 s the cars start moving and get up to speed. All cars start with $d_i(0) = 1\mathrm{m}$ and the desired distance is set to $D = 11\mathrm{m}$.

In the nominal case (no attack), all d_i converge to $11\,\mathrm{m}$ as expected, see Fig. 6.

With $A = -0.08$ (Fig. 7) we can see how car 1, shown as the orange solid line in the figure, gets closer and closer to the leader as d_1 approaches $0\,\mathrm{m}$, until it rear-ends it. Since we are not modeling the crash physics and assuming the two vehicles halt immediately, the behavior of car 2 and 3 after the crash is not meaningful; in this scenario the control law is not able to stop the car 2 and 3, causing a pile-up.

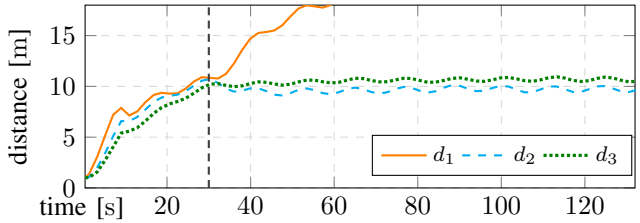


Figure 8: Distances between cars as time passes with $A > 0$

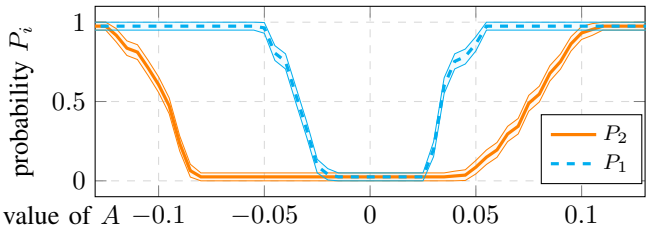


Figure 9: Probability of being out of interval within the first 10 seconds over A

Interestingly, with $A = +0.08$ (Fig. 8) we do not observe any crash; however car 1, shown as the orange solid line in the figure, tends to get further and further away from the leader, causing a loss in efficiency.

V. UPPAAL Probabilistic Queries

In this section we make use of UPPAAL SMC's queries.

A. For attack detection

Each car in the platoon tries to maintain a gap $\hat{d}$ to the vehicle in front as close as possible to desired distance D. During an attack such value might change quite drastically in a narrow window of time, making certain methods of detection unfeasible. We want to gauge the probability of the *error* being $\varepsilon_i = |\hat{d}_i - D| > 15\%$ during the first $10\,\mathrm{s}$ from the begin of attack, the lower the value ε_i the more time a hypothetical detector may have to properly identify the threat.

Assume the attack takes place on car 1 and the attack starts at $t_A = 30\mathrm{s}$. To check if the distance remains within margin between car 0 and car 1, and between car 1 and 2; the queries can be formulated as :

- `Pr[t_sim <= 40] (<> t_sim >= 30 && fabs((x_p[0] - x_p[1] - 4)/11 - 1) > 0.15)`
- `Pr[t_sim <= 40] (<> t_sim >= 30 && fabs((x_p[1] - x_p[2] - 4)/11 - 1) > 0.15)`

where each vehicle has length $4\,\mathrm{m}$.

Consider the first query, it estimates the probability of $\varepsilon_1 > 15\%$ at least once in the interval $[t_A, t_A + 10\mathrm{s}]$. The time interval is implemented by setting an upper bound of `t_sim <= 40` and by putting `t_sim >= 30` in conjunction with the formula to verify. The *at least once* is implemented by using the operator <>. The `fabs()` is the function to compute the *floating-point* absolute value.

We parametrized the value of A instead of stochastically assigning it at start-up. We varied the value of A and ran the queries. The results are shown in Fig. 9, the values of $P_i = P(\varepsilon_i \geq 0.15 \,|\, 30 \leq t \leq 40)$ are plotted against A, the confidence interval was set to 95%.

B. For safety

Let us now consider the problem of the safety of the overall platoon. That is, what's the probability, given an attack of amplitude A, of *not* having a significant effect on the distance between *all* cars in the first $10\,\mathrm{s}$ from the begin of the attack.

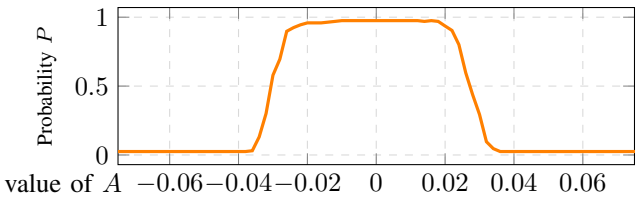


Figure 10: Probability of keeping the correct distance

This is the complementary problem of the one exposed in the previous section and can be formulated as the following UPPAAL SMC query:

```
Pr[t_sim <= 40] ([] t_sim >= 30 imply forall (i
: int[1, 3]) fabs((x_p[i-1] - x_p[i] - 4)/11 -
1) < 0.15)
```

The query thus formulated is quite similar to the one described in the previous section. In particular, it computes the probability of $\forall i\ \varepsilon_i < 0.15$ in the first ten seconds of an attack on car 1 with $t_A = 30$s. The `imply` operators implements the logical implication and it is used to check the condition on the error only when the antecedent $t \geq 30$s is true; when the antecedent is false the expression already evaluates to true. The operator `[]` means the condition must hold true continuously until the upper bound on `t_sim` is reached. The query is equivalent to state

$$P(\forall t \leq 40\text{s}, \quad t \geq 30\text{s} \implies \forall i \quad \varepsilon_i < 0.15).$$

We ran the query multiple times varying A in the same way as we did in the previous section.

The results are shown in Fig. 10, with confidence interval of 95%. We can see how very small values of A do not effect the safety.

The results align with those from the previous section, a conclusion that could not have been assumed a priori, given the uncertainty regarding the statistical independence of the operands in the conjunction $\forall i\ \varepsilon_i < 0.15$.

VI. Discussion and Conclusions

This work demonstrates UPPAAL's effectiveness in modeling and analysis of attacks in cyber-physical systems through: automata integrating continuous dynamics (vehicle kinematics) with discrete events (network delays and time-discrete controllers); automata with stochastic variables capable of modeling a wide range of behaviors; and probabilistic analysis of emergent behaviors via statistical model checking.

The results show several critical insights into the effects of attacks in the platoon.

The statistical model checking approach proved particularly effective for evaluating safety properties in cyber-physical systems. By formulating safety requirements as probabilistic queries, we can quantitatively assess certain behaviors of a system. Our probabilistic analysis provides foundation for intrusion detection systems. The probability $P(\varepsilon_i \geq 0.15)$ within 10-second windows serves as effective metric for real-time anomaly detection.

Our attack model focuses on steady-state sinusoidal perturbations. As further work, it might be interesting to consider more sophisticated time-varying attack patterns that could be employed by attackers.

References

[1] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *20th USENIX security symposium (USENIX Security 11)*, 2011.

[2] C. Bergenhem, S. Shladover, E. Coelingh, C. Englund, and S. Tsugawa, "Overview of platooning systems," in *Proceedings of the 19th ITS World Congress*, (Vienna, Austria), pp. 22–26, Oct. 2012.

[3] C. Quadri, V. Mancuso, M. A. Marsan, and G. P. Rossi, "Edge-based platoon control," *Computer Communications*, vol. 181, pp. 17–31, 2022.

[4] A. Virdis, G. Nardini, and G. Stea, "A framework for MEC-enabled platooning," in *IEEE WCNCW Workshop*, pp. 1–6, IEEE, 2019.

[5] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *Runtime Verification* (H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, eds.), (Berlin, Heidelberg), pp. 122–135, Springer Berlin Heidelberg, 2010.

[6] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[7] J.-P. Katoen, "The probabilistic model checking landscape," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, (New York, NY, USA), p. 31–45, Association for Computing Machinery, 2016.

[8] G. Behrmann, A. David, and K. G. Larsen, *A Tutorial on Uppaal*, p. 200–236. Berlin, Heidelberg: Springer, 2004.

[9] M. Barbier, A. Renzaglia, J. Quilbeuf, L. Rummelhard, A. Paigwar, C. Laugier, A. Legay, J. Ibañez-Guzmán, and O. Simonin, "Validation of perception and decision-making systems for autonomous driving via statistical model checking," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, p. 252–259, June 2019.

[10] C. Bernardeschi, G. Lettieri, and F. Rossi, "Statistical model checking of cooperative autonomous driving systems," in *Leveraging Applications of Formal Methods, Verification and Validation. Rigorous Engineering of Collective Adaptive Systems: 12th International Symposium, ISoLA 2024, Crete, Greece, October 27–31, 2024, Proceedings, Part II*, (Berlin, Heidelberg), p. 316–332, Springer-Verlag, 2024.

[11] R. Rajamani, H.-S. Tan, B. K. Law, and W.-B. Zhang, "Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 4, pp. 695–708, 2000.

[12] A. David, K. G. Larsen, A. Legay, M. Mikuăionis, and D. B. Poulsen, "Uppaal smc tutorial," *Int. J. Softw. Tools Technol. Transf.*, vol. 17, p. 397–415, Aug. 2015.