Soft Computing manuscript No.
(will be inserted by the editor)

Modeling and Analysis of Data Corruption Attacks and Energy Consumption Effects on Edge Servers using Concurrent Stochastic Games

Abdelhakim Baouya* · **Brahim Hamid** · **Levent Gürgen** · **Saddek Bensalem**

Abstract The intricate nature of modern edge architectures, relying on a vast array of computational logic and lightweight communication protocols, creates vulnerabilities that expose them to a broad spectrum of security threats. Moreover, security vulnerabilities can significantly impact the energy footprint of edge servers in these architectures. Our approach utilizes the Concurrent Stochastic Game (CSG) formalism to precisely model the behavior of IoT communication entities (players) while accounting for potential attacks at the communication edge and the resulting energy consumption caused by such attacks. We rely on the PRISM-games language for automated analysis where the game goals modeling functional and security requirements are expressed using reward Probabilistic Alternating Temporal Logic (rPATL). To validate our approach, we examine a data corruption attack applied to dam water flow control and study its side effect on energy consumption. Our key innovation lies in using formal models at the architectural level to explore potential attacks. These models capture synchronous and asynchronous communication styles, along with their associated energy consumption. The methodology and the implemented formalism offer a significant advancement over traditional game equation models while still achieving the desired security and energy evaluation. Numerical results show that compared to synchronous communication, asynchronous styles suffer from significantly larger infected buffers and higher energy consumption due to attacks ranging from 66% to 91%.

Keywords Edge Computing · Security Threats · Concurrent Stochastic Game · Temporal Logic.

*Abdelhakim Baouya
IRIT, Université de Toulouse, CNRS, UT2, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France
E-mail: abdelhakim.baouya@irit.fr

Brahim Hamid
IRIT, Université de Toulouse, CNRS, UT2, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France
E-mail: brahim.hamid@irit.fr

Levent Gürgen
Kentyou, France
E-mail: levent@kentyou.com

Saddek Bensalem
Université Grenoble Alpes, VERIMAG, CNRS, Grenoble, France
E-mail: saddek.bensalem@univ-grenoble-alpes.fr

Contents

1 Introduction

Multiple communication paradigms have been introduced to address the complexities of communication. The Cloud and Fog paradigms have emerged as solutions for handling massive amounts of incoming data [1, 2]. Cloud computing offers on-demand access to shared computing resources via the internet, enabling users to store, process, and manage their data and applications using remote servers and data centers [3]. It offers scalability to handle large demands; however, disruptions and latency resulting from multi-hop access can limit or hinder user access [4]. Fog computing [5] extends cloud computing capabilities to the network's edge, reducing latency [6]. Edge computing involves processing and analyzing data near its source or at the edge of a network [7]. Instead of sending all data to the cloud and data center for processing [8], edge computing brings computation and storage closer to where the data is generated. This approach allows for local data preprocessing, optimizing network performance while facilitating efficient decision-making at the edge of the network infrastructure [9]. These paradigms offer distinct benefits in handling communication complexities by effectively utilizing distributed resources while considering factors such as scalability, latency reduction, optimized network performance, and intelligent decision-making processes [10, 11]. Consequently, designers should leverage each paradigm to construct efficient architectures while also addressing security considerations at that level. For example, in an Edge-based architecture, designers must consider the presence of counterfeit devices and potential attacks that may exploit edge-supported protocols [12, 13].

Security concerns have become increasingly crucial in modern software-intensive systems, requiring early identification in the initial development stages and at the highest levels - particularly during the architecture design phase, where their semantics are most clearly defined [14]. For instance, architecture threat modeling and analysis is beneficial when it comes to detecting exploitable threats at the early stages of system development processes [15]. Looking at it from a different angle, software components' technology has become a key focus in research and development within software engineering due to its tremendous success in the market. This accomplishment is largely attributed to the crucial role that reuse plays [16]. Hence, edge-based modeling and analysis have to be considered at some point in component-based software development processes.

In order to enhance comprehension for both readers and non-specialists, it is crucial to grasp two fundamental aspects of security: threat models and attack models. For example, when engaging in threat modeling utilizing the STRIDE framework [17, 18], the primary focus lies in developing an understanding of the system's behavior (i.e., what is known), identifying vulnerabilities present within the system and suggesting potential solutions for mitigating these vulnerabilities. On the other hand, the attack model [19] revolves around the capabilities and strategies employed by an adversary to harm the system by exploiting system vulnerabilities, making use of their expertise and knowledge of the system (acquired through the analysis of threat models). It is worth noting that the system itself lacks awareness of the attacker's specific strategies. The attack model leverages the vulnerabilities identified within the threat model to execute successful attacks. In Fig. 1, we illustrate how the two models are interpreted. This example illustrates a malicious entity carrying out attacks on a system to gain unauthorized access to a resource and subsequently modify its content. The attacker model integrates both the threat model and the attacker's capabilities. This comprehensive understanding serves as the foundation for effective attack preparation strategies. Given this model, attackers are likely to exploit identified vulnerabilities, selecting the most efficient path to achieve their objectives.

Contribution overview To the best of our knowledge, the data corruption attacks [20] with energy effects and mitigation have not been previously addressed within the context of CSG formalism [21], specifically for both synchronous and asynchronous communication styles. In contrast to prior works that employed mathematical equation models [22, 23], this study leverages automata-based models for communication styles formulation. Furthermore, analysis using PRISM-games model-checking techniques has not been conducted before. Our proposed work aims to fill this gap by offering an accurate and early assessment of software architecture security, thereby safeguarding against exploitable threats while reducing the overall design effort required for edge-based architectures. In this regard, our contribution can be summarized in four key aspects:

1. Defining a formalism to specify a software architecture within component–port–connector fashion and communication styles [24] using CSG semantics,

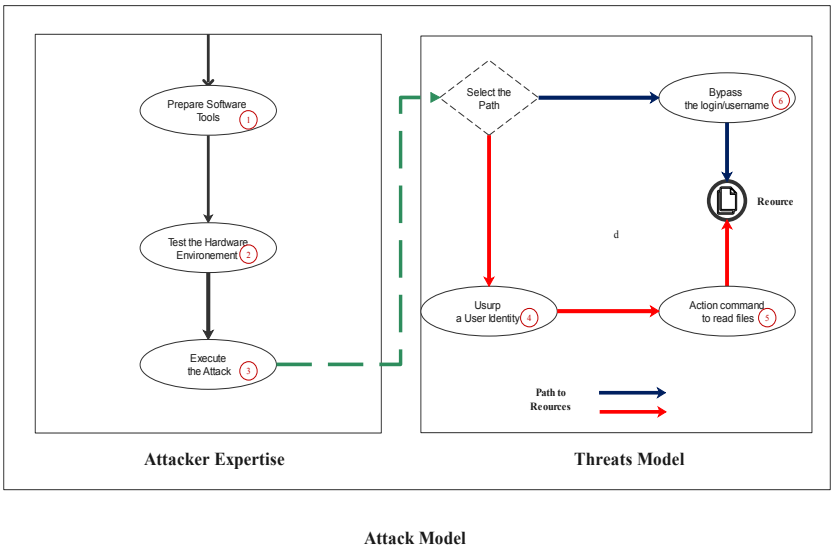

Fig. 1: Attack and Threats Model.

2. By establishing the transformation algorithm, we enable the conversion of the edge protocol language, *sensinact*, into the formalism of components, utilizing CSG semantics,
3. Formally express both functional and security requirements for the modeled system by defining its game goal using the rPATL specification language.
4. Evaluation of the two previous contributions in the context of water flow regulation- [25] using PRISM-games model checking.

2 Related Works

In this section, we delve into the research conducted in the context of implementing mathematical equations, Meta-Heuristics algorithms, and employing formal methods to address security concerns within the system architecture domain. Our review of related works focuses on three main aspects, as illustrated in Fig. 2.

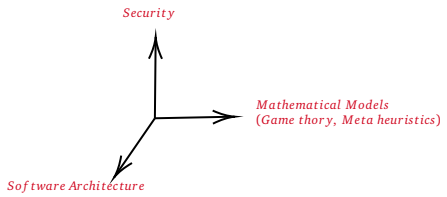

Fig. 2: Organization and Structure of Related Work Analysis.

2.1 Meta-Heuristics algorithms for security assessment and analyzing system vulnerabilities

Zhang et al. [26] propose a detection method based on neural networks deployed to the edge cloud. This approach enhances real-time performance, effectively detects attacks, and achieves higher de-

tection accuracy compared to existing methods. The authors emphasize the importance of setting appropriate learning parameters to ensure the model's accuracy is maximized. A multi-attack intrusion detection system (IDS) for edge-assisted IoT is proposed by Wu et al. [27] using the backpropagation (BP) Neural Network and Radial basis function (RBF) neural network. The BP neural network is utilized to identify abnormalities and assess the importance of features for each attack, while the RBF neural network is employed for detecting multi-attack intrusion. This work conducted by Wu et al. [28] addresses the challenges of edge computing in meeting low latency and ease of use requirements for IoT. It proposes an intelligent intrusion detection algorithm that combines generative adversarial network (GAN), a fuzzy rough set, and convolutional neural network (CNN) techniques. The algorithm performs feature selection using a fuzzy rough set, implements intrusion detection based on selected features using CNN, and achieves higher accuracy compared to existing methods. The work in Huang and Zhang [29] explores the use of edge computing for intelligent processing of image data generated by the Internet of Things (IoT). It establishes an edge agent using a shallow neural network deployed on edge devices and a cloud agent using a deep neural network deployed on cloud servers.

The multi-agent system combines edge and cloud cooperation to achieve fast local inference using convolutional neural networks and improve system performance accuracy by transmitting data to the cloud for additional processing using deep neural networks. The study by A et al. [30] proposes HetIoT-NIDS, a decentralized on-device misuse detection scheme that addresses offline misuse network IDS. HetIoT-NIDS is capable of efficient inference over smaller data samples and is deployable on low-memory and low-power IoT devices. Experimental results demonstrate the effectiveness and adaptability of this approach for online and offline intrusion detection, particularly with smaller data sample sizes. Despite the accuracy offered by applying metaheuristic algorithms, the computational cost of training and running neural networks can be significant, particularly for deep architectures with numerous layers and large datasets. This complexity poses practical limitations, especially in resource-constrained environments like Edge computing. Additionally, neural networks are vulnerable to adversarial attacks, which exploit their sensitivity to even minor input perturbations, thereby posing risks of incorrect predictions and compromised security systems. The analysis of security issues in software architecture is not adequately addressed using metaheuristic algorithms. Instead, it is more closely related to deployment decisions.

The works [31, 32, 33] address software deployment from the software architecture perspectives. Meedeniya et al. in [31] address the challenge of deploying software components to hardware nodes in embedded systems, particularly when the hardware architecture is designed before the software architecture. The proposed approach automates this task by targeting multi-criteria optimization and providing the system architect with near-optimal deployment alternatives that balance service reliabilities. The approach involves annotating the software and hardware architecture, quantifying deployment quality, and utilizing a Genetic Algorithm (GA) for implementation. Giannopoulou et al. in [32] address the challenges of deploying mixed-criticality applications on commercial multi-core platforms. They propose a comprehensive design flow, which includes a novel mixed-criticality aware model of computation and a correct-by-construction implementation approach. This design flow aims to enable the deployment of tasks that respond correctly to scheduling properties expressed in temporal logic, ensuring temporal isolation and certification requirements. Song et al. in [33] present a model-based approach for automatically assigning multiple software deployment plans to edge gateways and IoT devices in a distributed network. The approach utilizes constraint-solving techniques to assign deployment plans based on specific device contexts, enabling efficient and reliable software deployment. Security constraints are not considered in such deployment.

2.2 A game model for security assessment and analyzing system vulnerabilities

Eavesdropping attacks pose a significant challenge by causing interference in Unmanned Aerial Vehicles (UAVs) communication. To address these challenges, Kakkar et al. [34] propose a secure resource allocation scheme for UAV communication in wireless networks. Their approach leverages a zero-sum game theory framework and introduces a game model that optimizes resource allocation while maximizing the data rate and secrecy capacity of the communication channel. The research conducted by Chkirbene et al. [35] focuses on securing Wireless Sensor Networks (WSNs) against jamming attacks, which pose a significant threat due to the open nature of WSNs. The paper introduces an approach

based on the Colonel Blotto game model to minimize the worst-case impact of jamming attacks on sensor communications. By analyzing the Nash Equilibrium (NE) of the game and considering all potential attackers, an optimal power allocation strategy is computed to protect the network against malicious nodes. The paper authored by Chouikhi et al. [36] proposes a model that offloads computationally intensive tasks to edge and cloud servers while maximizing task completion before deadlines and minimizing energy consumption. To facilitate decision-making on task execution or offloading, a distributed cooperative game is introduced where each device considers task completion and energy consumption factors. The existence of Nash equilibrium is shown by proving the proposed game's status as a weighted potential game.

The research paper [37] examines the security concerns in cloud control systems (CCSs), specifically focusing on the threat of malicious false data injection (FDI) attacks that can impair system performance. The CCS defender is tasked with allocating defense budgets across nodes to safeguard the operation of CCSs. The interactions between the FDI attacker with the CCS defender are modeled as a Stackelberg game, then, optimal strategies for both parties are analyzed based on Nash equilibrium. The study [38] conducted by Hammar and Stadler demonstrates that game-theoretic modeling enables effective defender strategies against dynamic attackers, who adapt their tactics in response to the defender's actions. To obtain near-optimal defender strategies, an algorithm called Threshold Fictitious Self-Play (T-FP) is developed, which learns Nash equilibria through stochastic approximation. The research includes both simulation-based incremental learning of defender strategies and evaluation using an emulation system. This research work conducted by Borgo et al. [39] focuses on the security challenges faced by smart grids, which utilize information systems for enhanced energy distribution. The paper presents a game-theoretic model involving two consumers as active players capable of attacking and defending, along with a passive consumer. By framing the problem as a static game of complete information, theoretical and numerical analyses are conducted to explore Nash equilibrium solutions.

2.3 Security assessment from the perspective of formal methods

Formalizing threats for software architecture verification has been addressed in literature [24, 40, 14, 41, 42, 43] to study their consequences, document potential risks, and take steps to mitigate them. The work proposed by [24] introduces a metamodel that instantiates system architecture in the component-port-connector formalism which is then converted to Alloy. The formal architecture model is constructed based on predicates and assertions where components and connectors' behaviors are expressed as independent sets of action steps without automata logic. The feasibility of an assembly in the presence of threats can be determined using the Alloy analyzer. Meanwhile, authors in [14] model software architecture using BIP semantics within the component-port-connector formalism under the assumption that attackers are present through malicious components. However, it's worth noting that BIP connectors are stateless, which makes capturing connector-level threats challenging since behavior is expressed independently from automata logic [44]. In another approach described by authors in [43], software systems and CAPEC[45] threats were modeled using SysML with a Model-To-Model transformation mapping specifications into PRISM[46]. Probabilistic model checking was performed while specified probabilities depended on threat load corresponding with Kent's estimation[47] probability. However, SysML's level of expressiveness regarding data manipulation hinders its ability to handle complex models. The work in [48] models security concerns from the perspective of the RabbitMQ broker. This modeling targets the low-level representation of the communication broker considering one attacker, rather than focusing on addressing concerns at a higher, more abstract modeling level.

The authors in [42] capture a reduced graph of the attack model using Promela (Process Meta Language) [49], which pinpoints successful attacks. The model is then consumed by the SPIN model checker [49], where Linear Temporal Logic (LTL) formulas are used to specify properties expressing whether an attack has occurred or not and whether it violates correctness. However, it is important to note that the model may not fully capture the whole facets of the system architecture since it is guided by the intentions of the modeler.

Various approaches have been proposed for modeling threats in specific development contexts. In [41], threats are modeled as an Attack Tree (AT). The authors propose an algorithm to transform

this AT into a Markov Decision Process (MDP), which enables reasoning about spoofing properties using the PRISM model checker. Alternatively, work has also been done on modeling ATs in TSG - a two-player version of CSGs - where both attackers and defenders are considered as players.

2.4 Comparaison

We present detailed research activities focusing on developing models to address security concerns in Cyber-Physical Systems (CPS). Many existing research approaches [34, 35, 36, 37, 38, 39] rely on mathematical representations of the system, incorporating Meta-Heuristics algorithms [31, 32, 33] and game models that consider communicating entities such as WSNs, UAVs, and IoT devices. However, for designers who are not well-versed in mathematical modeling, this challenge can impede system production and deployment. Additionally, analyzing the model universe can be difficult due to the NP-Hard nature of model quality estimation. Therefore, it is crucial to consider the level of abstraction in such approaches. From a software architecture standpoint, communication styles often receive insufficient attention at the required abstraction level for assessing system feasibility. Previous research has utilized formalisms like BIP, and SysML to model computer-based systems. Although these formalisms describe connections through coordination ports using component-port-connector structures [24], they have limitations when it comes to stateless connectors as they are vulnerable to attacks. Compared to previous contribution in [16, 24, 50], the system architecture is modeled in Alloy and Event-B, allowing for analysis using dedicated tools. However, these modeling languages lack the flexibility of PRISM, which can reduce model size for verification when feasible. . To overcome this architectural limitation, we enhance the expressiveness of our models by annotating components as players and representing the behavior of components and connectors in an automata style. This enables us to formally express attacks and threats, analyzing the models using the PRISM-games engine, which generates corresponding game models using CSGs. Furthermore, designers must implement security goals in rPATL within these extended models. We aim to provide more accessible methods for addressing security concerns in CPS design processes for non-mathematical experts. Moreover, our approach offers scalability, providing valuable insights into the modeled system using PRISM-games model checker

3 Outline

This paper is structured as follows. Section 4 formally defines the problem statement we address in this work. Following this, Section 5 details the semantics of components and connectors, with a particular focus on the PRISM-games language constructs employed. In Section 6, we introduce the Sensinact gateway, a crucial component within our proposed framework. This section also elaborates on how threat manifestation is modeled using the PRISM formalism, with specific attention given to edge bridges. To demonstrate the effectiveness of our approach, Section 7 presents a practical implementation within the context of water flow regulation. Finally, Section 8 provides concluding remarks. This concluding section not only summarizes the key findings of this research but also outlines potential avenues for future exploration.

4 Problem Statement

Our work aims to utilize formal methods to specify and analyze security threats in Edge-based architecture accurately in the context of the Component-Port-Connector Architecture Model (CPC) [51] and Message Passing Communication System (MPS). We rely on the standard graphical representation of CPC elements and MPS characteristics for the specification and verification of functional and security requirements: 1) a Concurrent Stochastic Game (CSG) [52] formalism and reward Probabilistic Alternating Temporal Logic (rPATL) [53], as a technology-independent formalism; (2) formalization and verification using PRISM [46] as a tooled language used for modeling and analysis of component-based software systems.

This work is conducted within the context of a research project focused on model-based security and dependability. We are collaborating with smart Edge infrastructure suppliers, and our collaboration with *Kentyou*[1] has identified a need for this work. This infrastructure incorporates a routing mechanism that facilitates communication among different protocols. This article builds upon concepts from the European Project Brain-IoT [54]. Here, we present a novel approach for formally analyzing quantitative properties of threats and attacks, particularly *tampering*, in an Edge-based architecture using the Asynchronous Message-Passing Communication Style (AMPS). This threat can be exploited at the edge level and has the potential to disrupt decision-making processes at cloud/Fog servers.

To enhance the effectiveness of security architects, we propose integrating an explicit attacker model into the design stages. Unlike previous approaches that utilize unnamed components in counterexamples for threat detection (as proposed in [24]), our approach introduces a dedicated attacker component that directly represents the exploitation of vulnerabilities at the edge. We can determine whether these attacks succeed or fail by defining operational semantics that capture the resulting behavior. In order to represent this configuration accurately, we employ Concurrent Stochastic Games (CSG) as a suitable modeling technique for capturing concurrent access to the Edge, with each involved communicating entity represented as a player.

The workflow of our approach is depicted in Fig. 3. The model is constructed using components and connectors, with components representing various players such as sensors, actuators, or attackers. Connectors are responsible for establishing connections between these components through ports. The edge communication protocol, known as *sensinact*, is mapped into the component semantics as a player for architectural composition. To further enhance the model, attackers' frequency can be incorporated to analyze the impact of tampering (as part of STRIDE attacks). Finally, the model is fed into PRISM-games [21], a tool for model checking. PRISM checks the model against security goals expressed in rPATL, a property specification language. The output of this process can be quantitative (e.g., probability of attack success/failure) or qualitative (e.g., identification of security weaknesses).

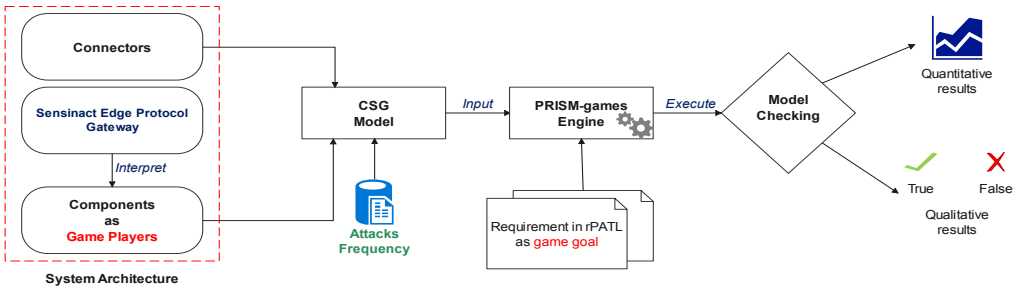

Fig. 3: Proposed Flow for Model Checking Components and Connectors using PRISM-games.

5 Modeling Components as Players in PRISM-games

Before diving into the theoretical concepts of components and connectors in PRISM, readers should consult the appendix-A on the preliminaries of Concurrent Stochastic Games (CSGs). In this section, we present the semantics of the component-port-connector (CPC) in PRISM as the target language, which offers support for the CPC concepts. We begin by providing definitions for components, connectors, and the CPC architecture. Subsequently, we outline the definition of a stochastic component as follows:

[1] http://kentyou.com/

Definition 1 (Component). A component in CPC is a PRISM module $\mathscr{D}_C = \langle s_0, S, P, C, \vartheta, L \rangle$ labeled with ports, where:

- s_0 is the initial state,
- $S = \{s_1, \ldots, s_k\}$ is a set of states,
- P is a set of ports referred to as PRISM actions in synchronized components,
- ϑ is a set of PRISM variables including state variables S ,
- $Cm : S \times P \times Const(\vartheta) \longrightarrow Dist(S)$ is a probabilistic PRISM command function assigning for each $s \in S$ and $p \in P$ a probabilistic distribution $\mu \in Dist(S)$, the behavior of a set of commands follows the Push, Pull, Update rules in Fig. 4 in the formalism of appendix A.3. θ is a set of valuations on a set of PRISM model variables ϑ , and
- $L : S \longrightarrow 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP).

Note : Authors and developers of PRISM-games model checker note that CSGs are handled differently [52, 21] as it is mentioned in [55] that: "*For a CSG, each player controls one or more modules and the actions that label commands in a player's modules must only be used by that player; this is a little different from PRISM's usual approach to components synchronizing on common action labels.*"

- **Push**. This axiom makes local data m available to other components with probability λ using port p :

$$\frac{s_i \xrightarrow{g_1 : p!m}_{\lambda} s_i' \wedge \theta \models g_1 \wedge s_j \xrightarrow{g_1 : p!m}_{1-\lambda} s_j' \wedge \theta \models g_2}{\langle s_0, \ldots, s_i, \ldots, s_n, \ \theta \rangle \xrightarrow{p}_{\lambda} \langle s_0, \ldots, s_i', \ldots, s_n, \ \theta' \rangle}$$

  where ϑ corresponds to state variables.

- **Pull**. This axiom illustrates how components read the value of m that is provided by external components through **Push**:

$$\frac{s_i \xrightarrow{g_1 : p?m}_{\lambda} s_i' \wedge \theta \models g_1 \wedge s_j \xrightarrow{g_1 : p?m}_{1-\lambda} s_j' \wedge \theta \models g_2}{\langle s_0, \ldots, s_i, \ldots, s_n, \ \theta \rangle \xrightarrow{p}_{\lambda} \langle s_0, \ldots, s_i', \ldots, s_n, \ \theta' \rangle}$$

  where ϑ corresponds to state variables.

- **Update**. This axiom describes the probabilistic updates for variable v_i related to s_i :

$$\frac{s_i \xrightarrow{g : p}_{\lambda} s_i' \wedge \theta \models g}{\langle s_0, \ldots, s_i, \ldots, s_n, \ \theta \rangle \xrightarrow{p}_{\lambda} \langle s_0, \ldots, s_i', \ldots, s_n, \ \theta' \rangle}$$

  where $\theta' = \theta[v_i := eval(v_i)]$

- **Composition**. This axiom permits the synchronization between components on a set of port p_1 , p_2 using the connector $\mathscr{D}_B$:

$$\frac{[\![\mathscr{D}_{C_1}]\!] = s_i \xrightarrow{g_1 : p_1} s_i' \wedge \theta_1 \models g_1 \wedge [\![\mathscr{D}_B]\!] = s_j \xrightarrow{g_3 : \langle p_1, p_2 \rangle} s_j' \wedge [\![\mathscr{D}_{C_2}]\!] = s_k \xrightarrow{g_2 :: p_2} s_k' \wedge \theta_2 \models g_2}{\langle s_0, \ldots, s_i, \ldots, s_j, \ldots, s_k, \ldots, s_n, \ \theta \rangle \xrightarrow{\langle p_1, p_2 \rangle} \langle s_0, \ldots, s_i', \ldots, s_j', \ldots, s_k', \ldots, s_n, \ \theta' \rangle}$$

  where $X' = X[v_i := eval(v_i)$ and $v_j := eval(v_j)]$

Fig. 4: Components and Connectors Operational Semantics Rules.

The semantics of connectors are derived from the semantics of components, with the connectors themselves not being considered as *players*. The connectors connect the components based on the component's ports as we use the operator γ . Expressing algebraically, the connections are modeled as composition such that $\gamma_{p_1, \ldots, p_n}(\mathscr{D}_{C_1}, \ldots, \mathscr{D}_{C_n})$ recorded bu connectors. So, Connectors are charac-

terized by a set of ports that label the commands within them. The formal definition of a connector is presented below:

Definition 2 (Connector). A connector $\mathscr{D}_B$ is a PRISM module such that $\mathscr{D}_B = \langle s_0, S, P_1 \times \ldots \times P_n, C, \vartheta, L \rangle$, where:

- s_0 is the initial state,
- $S = \{s_1, \ldots, s_k\}$ is a set of states,
- ϑ is aset of PRISM variables including state variables S ,
- $Cm : S \times P_1 \times \ldots \times P_n \times Const(\vartheta) \longrightarrow Dist(S)$ is a probabilistic PRISM command assigning for each $s \in S$ and a set of ports $p_1, \ldots, p_n \in P$ a probabilistic distribution $\mu \in Dist(S)$, the behavior of a set of commands follows the composition rule in Fig. 4, and
- $L : S \longrightarrow 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP).

Based on Definition 1 and Definition 2, we define a set of players $\mathscr{P}$, and γ as composition function. The CPC architecture is defined as follows:

Definition 3 (CPC Architecture). A CPC architecture is a game-based composition of components $\mathscr{D}_{C_1}, \ldots, \mathscr{D}_{C_n}$ with a connector $\mathscr{D}_B$ using $\gamma_{p_1,\ldots,p_n}(\mathscr{D}_{C_1}, \ldots, \mathscr{D}_{C_n})$ such that $\forall i \in \mathbb{N}\, \mathscr{D}_{C_i} \in \mathscr{P} \wedge \mathscr{D}_B \notin \mathscr{P}$

Example 5.1

The system depicted in Fig. 5 refers to the example in [48] and is portrayed in appendix A.2. In this context, the different actors of the example are treated as components and so as players, while connectors facilitate the scheduling and recording of push-pull operations. The resulting architecture is illustrated in Fig. 5. Three ports identify the first player: w_1 , r_1 and $reset_1$. Also, the second player is identified by three ports: w_2 , r_2 , and $reset_2$. The ports trigger the operation on the connector as modeled in Example A1. The connector model implements the operational semantics rule "Composition" as detailed in Fig. 4 whereas the players implement the operation rule "update".

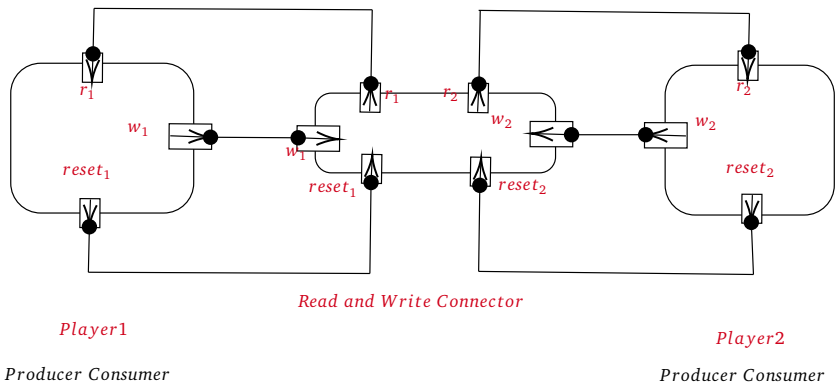

Fig. 5: Push and Pull Game Model in Component and Connector formalism.

The PRISM code in Listing 1 utilizes a connector to manage push and pull operations. All commands are labeled with two ports belonging to components (players) in charge of push and pull operations. The variable named "win" determines whether the player has been successfully pushed (1 if the first player won or 2 if the second player won the game). The initial commands on lines 5-6 represent unprocessed push or pull operations. When these operations are executed, a reset command becomes enabled on line 8, signaling an idle state. Following this, the commands on lines 10-11 schedule pushing and pulling operations. The model is available at [56].

Listing 1: PRISM Code for Recording Push/Pull on Connector Model of Fig. 5

```
module connector
        win : [0..2]init 0;
        a : [0..2]init 0;

        [w1,w2] s=0 -> (s'=1) & (win'=0);
        [r1,r2] s=0 -> (s'=1) & (win'=0);

        [reset1,reset2] s=1 | s=2 | s=3 -> (s'=0) & (win'=0);

        [r1,w2] s=0 -> (s'=2) & (win'=2);
        [w1,r2] s=0 -> (s'=3) & (win'=1);
endmodule
```

6 Edge-Supported IoT Architecture with SensiNact Gateway

In current IoT deployments, the IoT gateway serves as an interface between devices and high-level network domains is crucial. However, traditional gateway functionalities often remain limited to tasks such as traffic forwarding and protocol conversion [57, 58, 59]. To address these limitations, we leverage the capabilities of Sensinact, a smart gateway that offers advanced features. Sensinact allows end-users to customize routing protocols through a module called "sna". The utilization of the Sensinact gateway has been demonstrated in various projects, including BigClouT [60], WISE IoT [61], IoF2020 [62], ActivAge [63], and BRAIN-IoT [64]. As shown in Fig. 6, the architecture of Sensinact consists of two distinct modules: the Northbound module responsible for Internet communication and the Southbound module responsible for sensor communication.

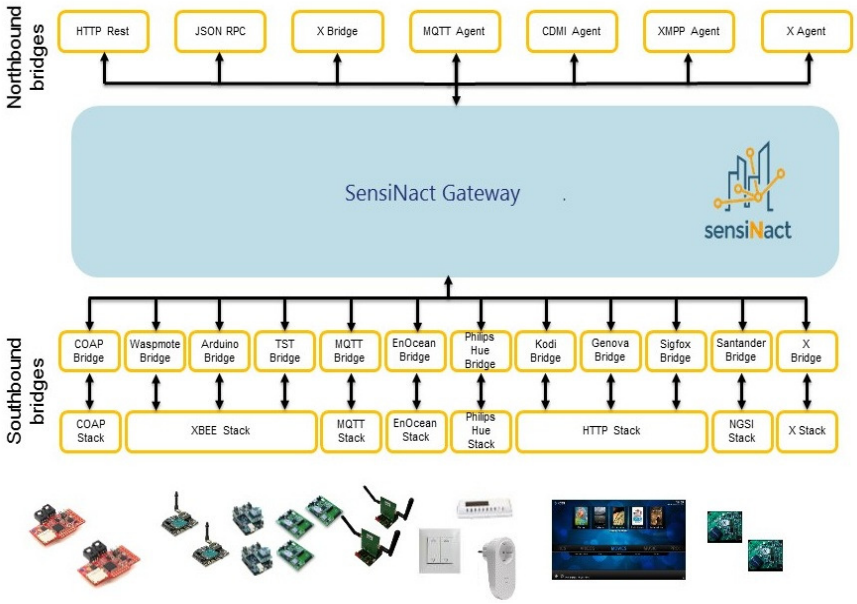

Fig. 6: Sensinact Gateway Internal Architecture [65].

- The *northbound* module provides functionalities for serving the gateway with remote end-user requests. It supports a wide range of protocols, such as HTTP REST, MQTT, XMPP, JSON RPC, and CDMI. Its primary purpose is to facilitate communication with remote servers.
- The *southbound* module facilitates the interaction with sensors and actuators, utilizing various device protocols such as Zigbee (for motion sensors), EnOcean (for remote controls, window opener detectors, etc.), LoRa, XBee, MQTT, XMPP, and CoAP.

6.1 Sensinact syntax

The execution mechanism of routing protocols in Sensinact follows the publish-subscribe pattern. Each protocol subscribes to the events of devices with an associated control structure behavior. The grammar of the protocol is illustrated in Listing 2.

Listing 2: An Overview of the Routing Protocol Grammar

```
1  [resource <[service]/[value]>]+
2
3  on <events>
4  [if <condition> do]+
5      [<actions>]+
6  end if;
```

The user defines, in the beginning, the exposed resources by each device using the keyword "**resource**". For instance, the **WL** device exposes "`WLValue`" service in Listing 3 (line 1). The protocol subscribes to the event that is triggered by the **Water Level**, so the subscription declaration shall be preceded by the keyword "**on**" (Listing 3, line 4) and concatenated to the event "**`.subscribe()`**". A control structure is defined after subscription as portrayed in Listing 3 (lines 5-10) by instantiating lines 4-8 of Listing 2. In line 6, the cloud water level value is set to the sensor value. To retrieve the values of services, the resource is concatenated to the keyword "**`.get()`**" as in the condition structure of Listing 3 (line 5). Hence, to trigger an action, the resource is concatenated to the keyword "**`.act()`**" as in Listing 3 (line 7).

Listing 3: Subscribing to WL events

```
1  resource WL=[WLtopic/wlValue]
2  resource cloud_wl=[cloudwl/cloud_wlValue]
3
4  on WL.subscribe()
5  if WL.get() > 10
6      cloud_wl.set(WL.get())
7      cloud_wl.act()
8  end if;
```

6.2 From sensinact to PRISM component

The transformation of Sensinact to CPC components is generally straightforward, with the exception of the edge bridges, which are treated as connectors. The instructions from the Edge are mapped to a set of commands within the game model CSG. However, it is important to note that special attention should be given to handling these edge bridges and ensuring their proper integration into the component-based architecture.

The algorithm presented in Algorithm 1 takes Sensinact instructions as input and generates the corresponding PRISM commands. Readers can visualize the structure of Sensinact as a tree, depicted in Fig. 7. The first level of nodes comprises a set of events, with each event containing the corresponding getters and setters instructions. To accomplish this transformation, the edge instruction *inst* is associated with three functions that facilitate: (1) collecting the event notification name using $inst.eventName()$, (2) obtaining the getterName through $getter.eventName()$, and (3) retrieving

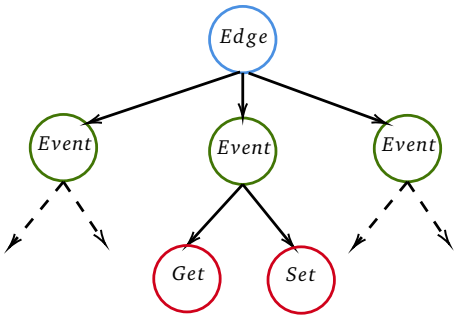

Fig. 7: Tree Node for Edge Instructions.

the setterName via $setter.eventName()$. The action performed over the edge will be mapped on "set" command, synchronizing it with northbound bridges and players responsible for handling such data.

Algorithm 1: Producing PRISM commands from Sensinact Edges.

Data: Edge model in Sensinact as E.
Result: Edge model in CPC.

1 $CPC \leftarrow \emptyset$; /* A list of PRISM commands. */
2 $Stack \leftarrow E$; /* A stack of Sensinact instructions. */
3 $index \leftarrow 0$; /* State location value. */
4 **Bool** $satify \leftarrow True$; /* Loop end verification. */
5 **for** $satisfy \neq false$ **do**
6 $inst \leftarrow POP(Stack)$; /* Retrieve Edge instruction. */
7 $Event \leftarrow inst.eventName()$; /* Retrieve the edge event. */
8 c_1= "[Event_SUBSCRIBE] s=index → (s'=s+1)"; ; /* Generate a subscription event. */
9 $index \leftarrow index + 1$
10 $GetEvent \leftarrow inst.getterName()$; /* Retrieve the event getter. */
11 c_2= "[GetEvent_GET] s=index → (s'=s+1) & (inst.ressource()'=BUFFER_inst.ressource())"; /* The command collect the southbound bridge buffer value. */
12 $index \leftarrow index + 1$
13 $SetEvent \leftarrow inst.setterName()$; /* Retrieve the event setter. */
14 c_3= "[SetEvent_SET] s=index → (s'=s+1)"; /* The command synchronizes with the northbound bridge to transmit the value. */
15 $index \leftarrow index + 1$
16 $CPC \leftarrow CPC \cup \{c_1\} \cup \{c_2\} \cup \{c_3\}$; /* Collect all the generated commands in one block. */
17 $index \leftarrow 0$
18 **if** $Stack\ is\ Empty$ **then**
19 $satisfy \leftarrow false$;
20 **end**
21 **end**

In accordance with Algorithm 1, the PRISM module is generated as shown in Listing 4. In this module, the input resource subscribed by the edge is converted into an integer value within the range [INIT_VAL..MAX_VAL] . The parameters INIT_VAL and MAX_VAL can be configured during model checking, providing flexibility. Additionally, the variable edge_loc represents the state of the components, where its maximum value is also adjustable based on designer preferences.

The algorithm instructions contribute to generating specific commands in lines 6-8 of the PRISM module. The first command (line 6) is produced by executing instructions from lines 7-8 of the algorithm. Similarly, the second command responsible for collecting input values (line 7) is generated using instructions from lines 10-11. Finally, instructions from lines 13-14 produce the last command found in line 8.

Listing 4: Generated Edge Component from Sensinact Code in Listing 3

```
module EdgeServer
WL: [INIT_VAL..MAX_VAL]  init -1;

edge_loc: [1..10] init 1;

[SENSOR_WL_SUBSCRIBE]   edge_loc=1  ->  (edge_loc'=2);
[SENSOR_WL_GET]   edge_loc=2  ->  (edge_loc'=3) & (WL'=BUFFER_WL) ;
[CLOUD_WL_SET]   edge_loc=3  ->  (edge_loc'=1);
endmodule
```

6.3 Communication and threats modeling at the edge

In the CSG model, the involvement of communicating entities is crucial. However, in the previous section, our focus was primarily on modeling the core edge independently from the edge bridges. In this section, we shift our attention to incorporate two communication styles implemented by Edge bridges that are captured by connectors within the CPC architecture.

Firstly, we present a model in Fig. 8, which includes three key players: Wl Sensor, Cloud, and Edge. Additionally, there are two connectors known as NorthBound and SouthBound bridges. While the implementation of WL_Sensor and Cloud is carried out by the architect (except for Edge, which follows the transformation discussed in the previous section), special emphasis is placed on designing connectors based on their respective communication style. Secondly, the attacker component, represented by the red box, is responsible for transmitting erroneous payloads through southbound bridges (i.e., connectors). Further details regarding malicious attacks will be addressed in the section on threat manifestation.

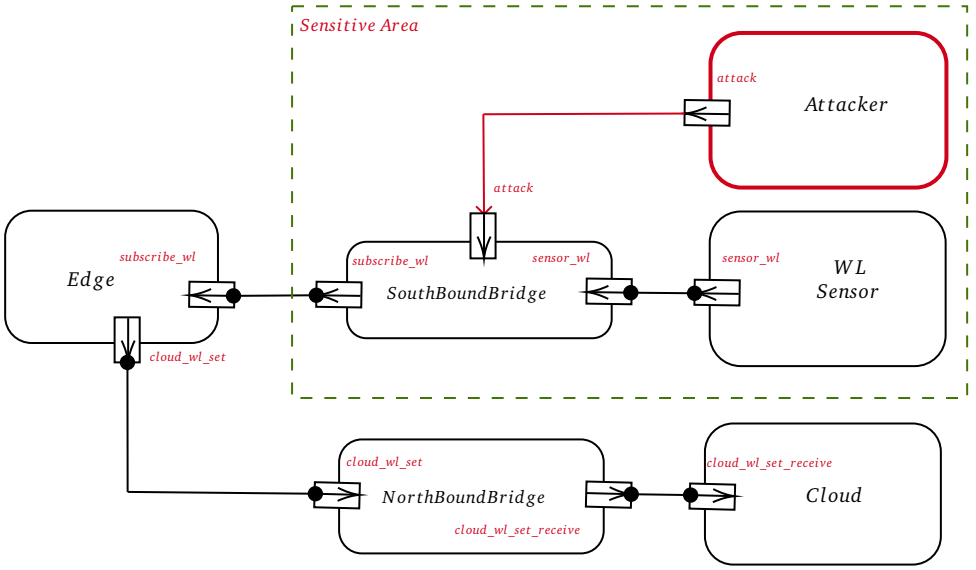

Fig. 8: Sensor-Edge-Cloud Architecture using Southbound and Northbound Bridges.

6.3.1 Attacks modeling

We explore tampering threats that impact the exchanged messages between communicating entities. [66] define tampering as the unauthorized modification of messages transmitted by the sender component through the communication port. In communication terminology, this translates to altering

the message payload during its transfer. In our specific scenario depicted in Fig. 8, these modifications are carried out by an entity known as the *attacker*. In the stochastic environment, the attacker operates at a specific frequency *freq* within both southbound and northbound bridges—two connectors responsible for transmitting data.

At the communication edge, the attacker can subscribe to specific topics at the bridges without notifying any new entry to the edge server. This situation implies a vulnerability in terms of payload security. The attacker can exploit such vulnerabilities, posing potential risks and threats. To better express successful or failed modifications, we use the operational semantics rules presented in Fig. 4. To illustrate these rules, we consider three players as components modeled in Fig. 8: $\mathscr{D}_{C_1}$ refers to the `WL_sensor`, $\mathscr{D}_{C_2}$ refers to `Edge`, and $\mathscr{D}_{C_3}$ to refer to the `Attacker`. The connector $\mathscr{D}_B$ is the southbound bridge responsible for synchronizing sensor events. `wl_pl` is the sensed data belonging to the sensor `WL_sensor`, whereas v_x is the value used to tamper the collected sensor data with frequency λ.

Based on the description provided, each component is represented by a set of commands. Each command is labeled with a specific port, except for the connector where the command is labeled with a set of ports $P = \{p_1, p_2, p_3\}$. The successful attack is modeled using the operational rule **Success**. To make correspondence with the architecture in Fig. 8, we use simple port names to reduce the size of the rule as $p_1 = sensor\_wl$, $p_2 = subscribe\_wl$, and $p_3 = attack$

$$\frac{\begin{array}{r}[\![\mathscr{D}_{C_1}]\!] = s_1 \xrightarrow{g_1:p_1!wl\_pl} s_1' \wedge [\![\mathscr{D}_{C_2}]\!] = s_2 \xrightarrow{g_2:p_2} s_2' \wedge [\![\mathscr{D}_{C_3}]\!] = s_3 \xrightarrow{g_3:p_3!v_x} s_3' \wedge \\ [\![\mathscr{D}_B]\!] = s_b \xrightarrow{g_b:\langle p_1,p_2,p_3?v_x\rangle}_{\lambda} s_b' \wedge\ \theta \models g_1 \wedge \ldots \wedge g_3\end{array}}{\langle s_1,\ldots,s_2,\ldots s_3,\ldots s_b,\ldots\theta\rangle \xrightarrow{\langle p_1,p_2,p_3\rangle}_{\lambda} \langle s_1',\ldots,s_2',\ldots,s_3',\ldots s_b',\ldots\theta'\rangle} \quad \text{(Success)}$$

where $\theta' := \theta[buffer\_wl := v_x]$

In the event of failure attacks, the `buffer_wl` remains untampered as the component `WL_Sensor` transmits data through the port `sensor_wl`. In the depicted architecture shown in Fig. 8, failures occur with a frequency of $1-\lambda$ on operational semantics **Failure**.

$$\frac{\begin{array}{r}[\![\mathscr{D}_{C_1}]\!] = l_1 \xrightarrow{g_1:p_1!wl\_pl} l_1' \wedge [\![\mathscr{D}_{C_2}]\!] = l_2 \xrightarrow{g_2:p_2} l_2' \wedge [\![\mathscr{D}_{C_3}]\!] = l_3 \xrightarrow{g_3:p_3!v_x} l_3' \wedge \\ [\![\mathscr{D}_B]\!] = s_b \xrightarrow{g_b:\langle p_1?wl\_pl,p_2,p_3\rangle}_{1-\lambda} s_b' \wedge\ \theta \models g_1 \wedge \ldots \wedge g_3\end{array}}{\langle s_1,\ldots,s_2,\ldots s_3,\ldots s_b,\ldots\theta\rangle \xrightarrow{\langle p_1,p_2,p_3\rangle}_{1-\lambda} \langle s_1',\ldots,s_2',\ldots,s_3',\ldots s_b',\ldots\theta'\rangle} \quad \text{(Failure)}$$

where $\theta' := \theta[buffer\_wl := wl\_pl]$

To enhance the understanding of the operational semantics rules for success and failure attacks, we employ the synchronous connector model from Listing 5 and introduce the attacker with the `sensor_at_pl` label in the command specified in line 6. The variable 'freq' represents the frequency of attacks on the connector edge, denoted by λ in the operational semantics. The success rule is satisfied when the attacks are triggered with a frequency of `freq`, whereas the failure rule is satisfied when the attacks are triggered. Still, the water level payload is transmitted with a frequency of `1-freq`.

6.3.2 Synchronous communication style

Synchronous communication is a mode of communication that necessitates explicit synchronization between the sender and receiver. In this type of communication, the sender typically waits for the receiver's confirmation or acknowledgment, indicating the payload (i.e., message) successful receipt or processing. The PRISM language offers constructs to facilitate such synchronization, as discussed in Fig. 4 of Section 5. Additionally, connectors play a crucial role in ensuring reliable data transmission, serving as bridges for northbound and southbound interfaces. Modeling such communication style is portrayed in Listing 5. Two variables `BUFFER_WL` identify the model initialized to

EMPTY = −1 with values within the range [INIT_VAL..MAX_VAL] (line 3) and CN1 that are still constant at value 1 (line 4). The PRISM command in line 6 synchronizes between WL_Sensor and the Attacker on both components ports SENSOR_WL_R and SENSOR_AT_R to model a stochastic attack and to notify the connector of the data reception. The second command facilitates synchronization between the sensor and the edge, allowing for the retrieval of sensed data stored in the buffer. Both commands fully implement the **Composition** rule. The implementation of the northbound bridge adheres to the same policy as the southbound bridge. Graphically, the communication automata is represented by two states (CN1 values) in Fig. 9.

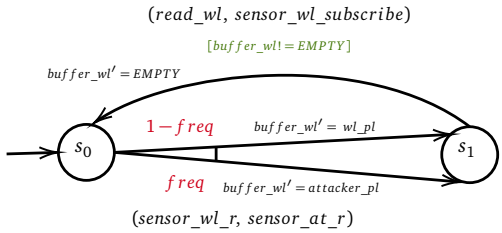

Fig. 9: Synchronous Southbound Edge for Water Level Sensor under Attack.

Listing 5: Synchronous Southbound Connector Model for Fig. 8

```
1  //SoutBoundEdgeBridge Connector implemented as PRISM module
2  module SouthBoundEdgeBridge
3    BUFFER_WL: [INIT_VAL..MAX_VAL]  init EMPTY;
4    CN1: [1..2] init 1;
5
6  [SENSOR_WL_R ,SENSOR_AT_R]    CN1=1  ->  (1-freq):(CN1'=2) &(BUFFER_WL'= wl_pl) + freq
       :(CN1'=2) & (BUFFER_WL'=ATTACKER_PL);
7  [READ_WL,SENSOR_WL_SUBSCRIBE] CN1=2 & BUFFER_WL!=EMPTY -> (CN1'=1)&(BUFFER_WL'= EMPTY)
       ;
8  endmodule
```

6.3.3 Asynchronous communication style

Asynchronous communication style refers to a mode of communication where the sender and receiver do not need to be engaged in simultaneous interaction. In this style, the sender can initiate a request and continue with another request without waiting for an immediate response from the receiver. The receiver, on the other hand, processes the message independently and responds at its own pace. This form of communication is referred to as Message Passing Communication System (MPS), as detailed in the work by [24]. The MPS style employs an array of buffers for writing message payloads when the buffer is empty, while the buffer values are consumed asynchronously by the cloud component at the edge. Modeling such communication style is portrayed in Listing 6. Three variables buffer_0_wl and buffer_1_wl are initialized to EMPTY with values within the range [INIT_VAL..MAX_VAL] that stores the first and the second incoming WL (lines 2-3) and CN1 that takes values in [1..2] (line 6) which represent the state values. The PRISM command in line 9 synchronizes between WL_Sensor and the Attacker on both components ports sensor_wl_r and sensor_at_r to make the connector ready for reading sensed data. In this case, we make use of the index of the available buffer to store the sensed data. The operation is managed using PRISM conditional expressions, as described in lines 11-12 and 14-15. Once the sensed data is stored in one of the buffer variables, it becomes ready for consumption by the Edge based on the index value specified in line 19. If a buffer at a specific index is consumed, the corresponding buffer variable is reset to an empty state. Both commands implement the **Composition**

rule. However, it is important to note that PRISM does not support arrays. To address this limitation, we model two buffers which are declared in lines 4-5 of the code snippet. Furthermore, the update of `buffer_0_wl` and `buffer_1_wl` depends on both the index value and the state of the buffer (whether it is empty or not). The index also increments based on the buffer size and current index value using modulo operations.

Listing 6: Asynchronous Southbound Connector Model for Fig. 8

```
1   //SoutBoundEdgeBridge Asynchronous Connector implemented as PRISM module
2  module SouthBoundEdgeBridge
3   BUFFER_WL   : [INIT_VAL..MAX_VAL]  init EMPTY;
4   BUFFER_0_WL: [INIT_VAL..MAX_VAL]  init EMPTY;
5   BUFFER_1_WL: [INIT_VAL..MAX_VAL]  init EMPTY;
6   INDEX_WL    :[0..2] init 1;
7
8  CN1: [1..2] init 1;
9  [SENSOR_WL_R ,SENSOR_AT_R]    CN1=1  ->
10 (1-freq):(CN1'=2) &
11     (BUFFER_0_WL'=(INDEX_WL=0)? wl_pl:EMPTY)&
12     (BUFFER_1_WL'=(INDEX_WL=1)? wl_pl:EMPTY)
13 + freq:(CN1'=2) &
14     (BUFFER_0_WL'=(INDEX_WL=0)? ATTACKER_PL:EMPTY)&
15     (BUFFER_1_WL'=(INDEX_WL=1)? ATTACKER_PL:EMPTY);
16
17 [READ_WL,SENSOR_WL_SUBSCRIBE] CN1=2 & (BUFFER_0_WL!=EMPTY|BUFFER_1_WL!=EMPTY) ->
18 (CN1'=1)&
19     (BUFFER_WL'= (INDEX_WL=0)? BUFFER_0_WL:BUFFER_1_WL)&(INDEX_WL'=mod(INDEX_WL,2))&
20     (BUFFER_0_WL'= (INDEX_WL=0)? EMPTY:BUFFER_0_WL)&
21     (BUFFER_1_WL'= (INDEX_WL=1)? EMPTY:BUFFER_1_WL);
22 endmodule
```

7 Industrial Use Case

The use case scenario is derived from a successful project in collaboration with 12 European partners: BRAIN-IOT [54]. The industrial case study involves the Cecebre dam infrastructure located in Spain. The primary objective of this dam is to collect water from the Meirama Lake and store it in the Cecebre Reservoir, which boasts a substantial total capacity of $146 \times 10^6 m^3$ and is situated approximately 20 km away from the city of La Coruña, this reservoir plays a vital role in meeting water supply demands for various purposes. In order to monitor and manage crucial aspects of the dam's operations, we have deployed advanced SICA[2] wireless sensor nodes throughout its premises. These device nodes collect and transmit three measurements (Similar to the device described in [67]) such as water level (WL), rain precipitation (RP), and water volume (WV) to the centralized SICA Edge system. Then, water flow is calculated through these sensors, and human operators effectively intervene to oversee spill gate management and control drainage operations at strategic points within the dam infrastructure. The system architecture for the industrial use case is illustrated in Fig. 10. The system consists of three sensors, one actuator, an edge node represented by the Sensinact gateway, and a fog/cloud node responsible for processing incoming data. The Sensinact gateway processes and routes the data to the devices using the MQTT protocol or forwards it to the cloud via HTTP requests.

7.1 System and game goal modeling

To represent the software architecture of the system depicted in Fig. 10, we utilize the graphical constructs of components and connectors as described in Section 5. It is important to note that we also consider the inclusion of edge connectors, specifically the southbound and northbound bridges, which are modeled in Section 6.3. The architecture model is illustrated in Figure [fig:architecture:latex]. It consists of seven components: (1) WL_Sensor as player p_1 , (2) RP_Sensor as player p_2 , (3)

[2] Sistema Integral del Ciclo del Agua (Integral System of Water Cycle)

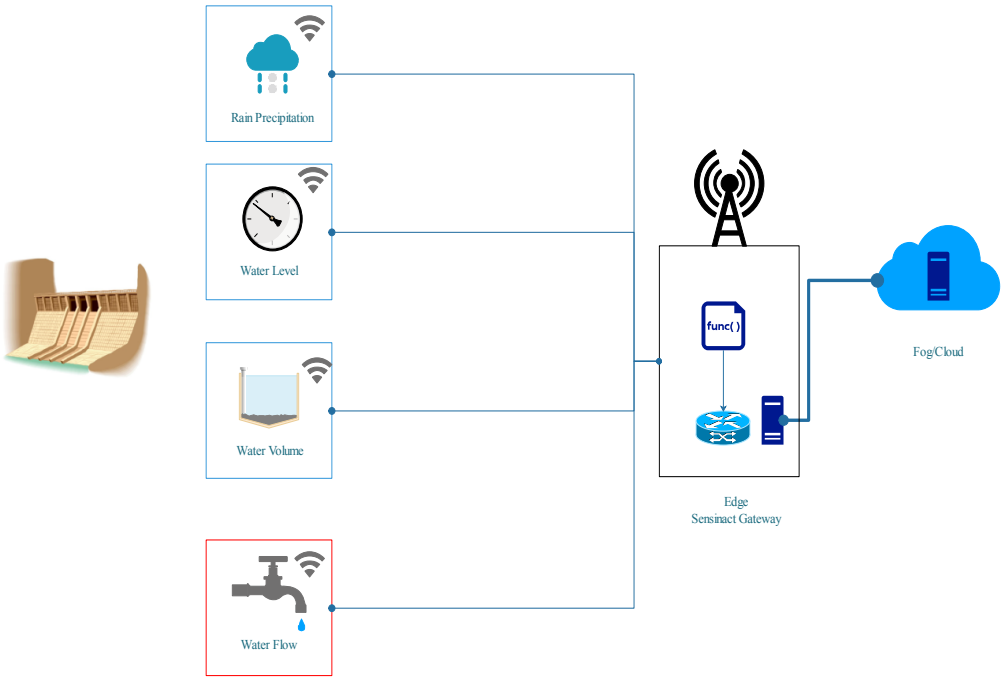

Fig. 10: Sensor-Edge-Cloud Architecture for IoT Systems.

WV_Sensor as player p_3 , (4) Edge as player p_4 , (5) WF_Actuator as player p_5 , and (6) Cloud as player p_6 . The connectors correspond to those previously modeled as Northbound bridges and Southbound Bridges. Additionally, an attacker as player p_7 is represented by a component that sends malicious data to corrupt the water flow prediction. Informally, we will focus on outlining the functional and security communication requirements that the software system under development must meet in order to ensure its effectiveness. *A complete presentation of our PRISM model is available online via* [56].

Experimental setup. Within the set of functional and security properties, we have encoded properties in rPATL formalism. PRISM-games model checker 3.0 [21] is utilized to perform verification. These experiments were conducted on a Ubuntu-I7 system equipped with 32GB RAM. Multiple engines can be selected (refer to documentation [68]) offering performance benefits for specific model structures. In addition, we have implemented the scenarios outlined in [69] to accurately model attack frequencies.

In scenario 1:

- Small chance an attack is successful (*attack*=0.2);
- High chance of payload delivery if the attack is prevented(*delivery*=0.8);

and in scenario 2:

- High chance an attack is successful (*attack*=0.8);
- Small chance of payload delivery if the attack is prevented(*delivery*=0.2);

PRO1: *When the sensors send payload messages to the cloud, what is the probability that the sensed values collected at the edge are tampered with, denoted as (WV, WL, and RP* $\in [1 \ldots 3]$*) within both scenarios at a certain number of rounds.*:

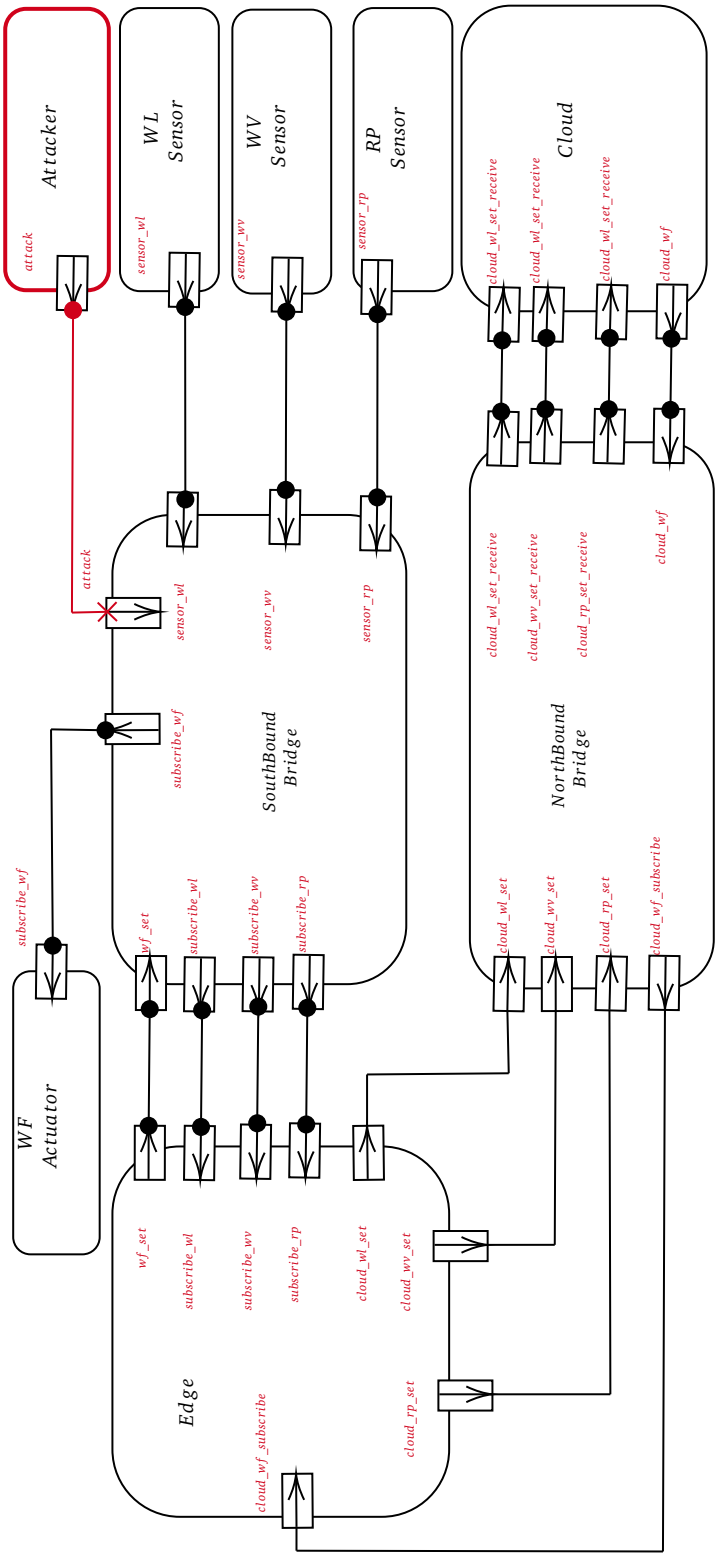

Fig. 11: Component-port-Connector Architecture of Physical System in Fig. 10.

$$P =?[(\texttt{WL} = \texttt{EMPTY} | \texttt{WV} = \texttt{EMPTY} | \texttt{RP} = \texttt{EMPTY}) U (\texttt{WL} = \texttt{ATTACK\_VAL} | \texttt{WV} = \texttt{ATTACK\_VAL} | \texttt{RP} = \texttt{ATTACK\_VAL})], \texttt{scenario} = 1:2:1, \texttt{rounds} = 1:30:1 \quad (1)$$

PRO2: *When the sensors send payload messages to the cloud, what is the expected number of messages that are tampered with at each round?:*

$$R\{\text{“}\texttt{incorrect}\text{”}\}\texttt{max} =?[\texttt{r} = \texttt{rounds}], \texttt{scenario} = 1:2:1, \texttt{rounds} = 1:30:1 \quad (2)$$

7.2 Experiments and analysis of the results

Artefacts. The source code for the experiments described in this section is publicly available on a GitHub repository[56]. The website provides comprehensive instructions on how to replicate the experiments. Furthermore, the repository includes a Python code that extracts attack frequencies to populate the PRISM model in PRISM. This research utilizes a dataset provided by the Canadian Institute for Cybersecurity.

7.2.1 Security risks assessment

Our initial experiment investigates the probability of tampering with a received payload containing sensed values, utilizing the “until” quantifier as described in RQ 1. This property was examined through a PRISM model in both synchronous and asynchronous communication modes after 30 rounds. The results, illustrated in Fig. 12 and Fig. 13, demonstrate that the probability of tampering with sensed data increases after 30 rounds in both scenarios. Additionally, a higher attack frequency leads to a tampering probability approaching 1. Notably, in the asynchronous mode, tampering tends to occur later compared to the synchronous mode, which can be attributed to potential delays in the consumption of buffers.

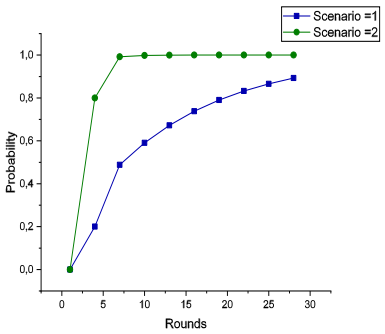

Fig. 12: Verif. Property 1 in Synch.Mode.

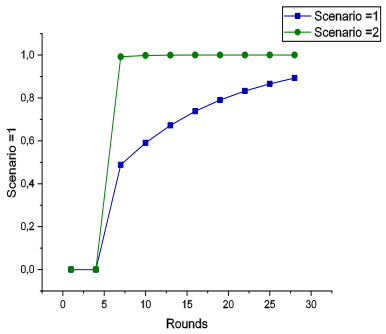

Fig. 13: Verif. Property 1 in Asynch.Mode.

The second experiment aims to investigate the expected number of incorrect payloads received at the buffer (with a size of 2) in both scenarios. To address REQ 2, we utilized PRISM-games and conducted the experiment over 30 rounds. The game goal involved a reward structure that encompassed all players (i.e., components), as depicted in the code snippet shown in Listing 7. Notably, the reward structure is synchronized in terms of command actions, with the second command specified

Listing 7: Reward Structure in PRISM Model

```
1  //incorrect messages during attacks
2  rewards "incorrect"
3
4          [READ_WV,SENSOR_WV_SUBSCRIBE] BUFFER_WV=ATTACKER_PL: 1;
5          [READ_RP,SENSOR_RP_SUBSCRIBE] BUFFER_RP=ATTACKER_PL: 1;
6          [READ_WL,SENSOR_WL_SUBSCRIBE] BUFFER_WL=ATTACKER_PL: 1;
7
8  endrewards
```

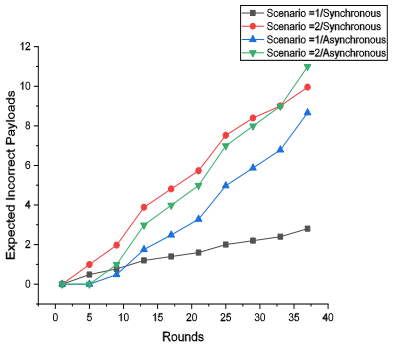

Fig. 14: Verif. Property 2.

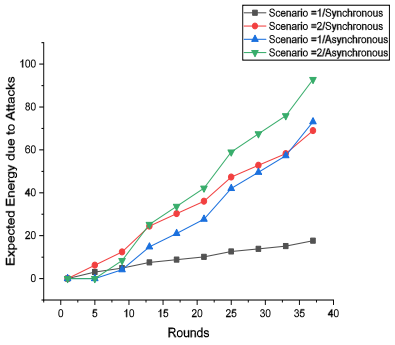

Fig. 15: Verif. Property 4.

in line 7 of Listing 5. In both communication modes depicted in Fig. 14, it was observed that as the frequency of attacks increased, the cumulative count of incorrect payloads also increased. It is worth noting that there is a notable difference between the communication modes in scenario 1, which can be attributed to the buffer size. However, in the second scenario, there is a slight difference between the synchronous and asynchronous modes.

7.2.2 Energy consumption and scalability

The third experiment in our research paper focuses on calculating the cumulative energy consumption using the methodology proposed by Kesrouani et al. in their work [70]. The formula for calculating energy consumption is expressed as:

$$E = 2.1514u + 4.142(W) \tag{3}$$

In this formula, the variable u represents the number of CPU cycles. To enhance the synchronous model described in the previous section, we introduce a new reward structure called `energy`, which is illustrated in Listing Listing 8. This reward structure allows us to calculate the energy consumption in the presence of attacks at the southbound bridges (i.e., connector). This assessment directly addresses the research question *RQ3*. By incorporating the `energy` reward structure, we gain valuable insights into the impact of attacks on energy consumption.

Listing 8: Reward Structure for Energy Consumtion in PRISM Model

```
1   //Modeling energy consumption in the absence of attacks
2  rewards "energy"
3          [READ_WV,SENSOR_WV_SUBSCRIBE] BUFFER_WV!=ATTACKER_PL: 2.1514 * u + 4.142;
4          [READ_RP,SENSOR_RP_SUBSCRIBE] BUFFER_RP!=ATTACKER_PL: 2.1514 * u + 4.142;
5          [READ_WL,SENSOR_WL_SUBSCRIBE] BUFFER_WL!=ATTACKER_PL: 2.1514 * u + 4.142;
6  endrewards
```

PRO3: *What is the expected energy consumption under tampering attacks within 30 rounds? This question should be addressed for both scenarios: high attack frequency and low attack frequency*:

$$\texttt{R\{“energy”\}max =?[r = rounds], scenario = 1 : 2 : 1, rounds = 1 : 20 : 1} \quad (4)$$

The experiments investigated the impact of tampering attacks on energy consumption for Property 3 (PRO3) in both synchronous and asynchronous communication modes. Table 1 presents the results in 30 test rounds. In asynchronous mode, the buffer size was set to 4 items. Here, the number of processor cycles required also depends on the number of corrupted items in the buffer. This is because the southbound bridges need to search for empty spaces to append data, increasing the workload.

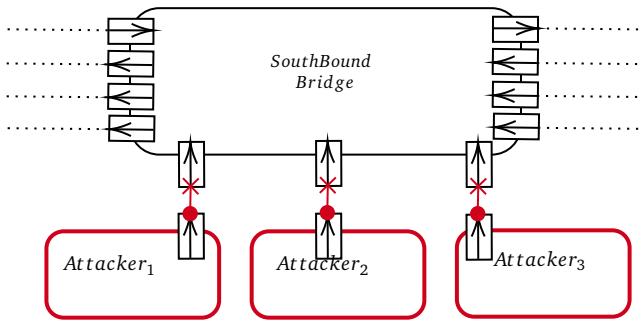

Fig. 16: A part of Component-port-Connector Architecture in Fig. 10 under Multiple Attacks.

We further extended the model by introducing additional attackers (see Fig. 16) to explore the relationship between attack frequency and energy consumption. The experiments assessed scenarios with one, two, and four attackers. In each experiment, we calculated the time required for verification (denoted as VT).

The results clearly show that energy consumption for tampering increases as the number of attacks rises. This poses a significant challenge for Raspberry Pi performance, as malicious behavior can reduce the edge device's lifespan due to increased processing demands. By analyzing this impact, researchers and practitioners can create strategies to mitigate these negative effects, ensuring the longevity and reliability of edge devices as in [64]. Furthermore, we can estimate the relative risk of attacks between synchronous and asynchronous communication styles by analyzing their energy consumption. To achieve this, we calculate the ratio of the mean energy consumption for Scenario 1 and Scenario 2. The mean energy consumption for Scenario 1 is 66%, while Scenario 2 exhibits a significantly higher value of nearly 91%. Consequently, the high attack rate in Scenario 2 leads to considerably higher energy consumption, which can hinder edge device operation.

NB. Attackers	Synchronous Communication Style				Asynchronous Communication Style			
	Scenario 1 (W)	VT (s)	Scenario 2 (W)	VT (s)	Scenario 1 (W)	VT (s)	Scenario 2 (W)	VT (s)
1	56.6406	0.005 s	226.56	0.008 s	85.68	0.025 s	248.19	0.022 s
2	113.281	0.014 s	453.12	0.021 s	171.36	0.103 s	496.38	0.103 s
4	226.56	0.03 s	906.24	0.034 s	342.73	6.853 s	992.76	6.946 s

Table 1: Verif. PRO3 in Sync. Mode and Async. Mode.

Formal verification often faces a significant challenge known as state space explosion. In PRISM models, for example, variables have defined ranges with specified maximum values to help mitigate this issue. However, the verification complexity still grows exponentially as the number of variables increases. This can be observed in the verification of PRO3. While the synchronous mode with four attackers completes in a mere 0.0034 seconds, the asynchronous mode takes significantly longer, requiring approximately 6.8-6.9 seconds – nearly double the time (see Fig. 17).The observed verification time (VT) increases significantly with larger buffer sizes. When verifying buffers exceeding 10 items, none of the models could be constructed in under 10 minutes. This suggests a limitation in the system's handling of large buffers. The system requests an increase in Java heap size to accommodate the additional memory requirements of processing larger buffers in asynchronous mode.

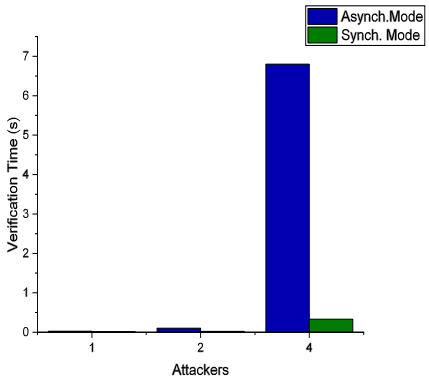

Fig. 17: Verification Time of PRO3.

Our experiment focused on the verification process, excluding model construction. Users can leverage models from [56] to observe the parameters associated with verification. Constructing verification models can be time-consuming, especially when identifying the numerous states and transitions needed for reachability matrices. This challenge intensifies with models involving two or more attackers. However, the verification process itself scales more favorably than construction time. Verification only requires path identification within the game model matrix. Abstraction and remodeling techniques can further mitigate state space explosion. For example, instead of modeling the entire architecture (as in our models), we can model only edge interfaces when verifying tampering-related properties. This approach reduces module size while effectively preserving the ability to verify connector security properties.

7.2.3 Mitigating architectural threats for reduced energy consumption

From a security standpoint, it is crucial to acknowledge that the edge architecture is highly susceptible to attacks. However, users must proactively take steps to mitigate the impact of these attacks on the architecture, aiming to increase the lifetime of the architecture and the physical infrastructure. An example of such mitigation measures can be seen as banning attackers [71] when it has been detected or moving attackers to decoy network [72] and cannot access the real network, thereby discouraging further malicious activities.

Implementing measures like attacker banning can significantly enhance the security of the edge architecture. This not only improves security but also reduces energy consumption by preventing malicious activity (pushing malicious payloads). To achieve this, the attacker model of Listing 9 has been updated by adding commands that block attacks at line 18 of the Listing 9 code block. Blocking is triggered when the `block` variable is set to true by the command in line 16, which implements attacker port blocking. While attack mitigation might be limited at the connector level, attacker behavior can be updated to enable effective mitigation strategies. A potential mitigation approach at the connector level could then involve dynamically adjusting the frequency variables to minimize the likelihood of attacks.

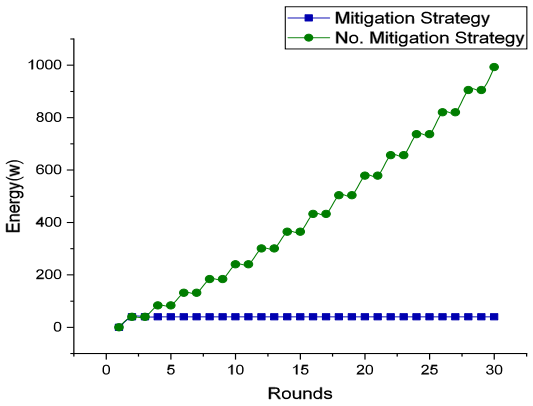

Fig. 18: Effects of Attacks Mitigation by Port Blocking in Asyn.Mode.

The results in Fig. 18 demonstrate the energy consumption impact of attack mitigation via port blocking in the case of asynchronous communication mode (The energy consumed is ****). Analyzing the green line, we see that energy consumption steadily increases as the model rounds grow, reaching over 992 watts. Conversely, with mitigation in place (represented by the blue line), energy consumption remains stable at around 40 watts, reflecting the effectiveness of port-blocking strategies.

Listing 9: Modeling Defense Strategy to Save Energy

```
1   //Defender player
2  module Attacker
3    //// Attacker payload
4    ATTACKER_PL: [INIT_VAL..MAX_VAL] init ATTACK_VAL;
5
6     // Tampering state (true/false)
7    tamper: bool init false;
8
9     // blocking port
10   block: bool init false;
11
12    // Active state (true/false)
13   active: bool init false;
14
15    // Initial activation (INIT_AT)
16   [INIT_AT] active=false & block=false-> (1-freq): (tamper'=false) & (active'=true) &
          (block'=true) + freq: (tamper'=true) & (active'=true)& (block'=true);
17    // Alternative command while attack is blocked
18   [INIT_AT_ALT] active=false & block=true -> (tamper'=false);
19    // Sensor reading (SENSOR_AT_R) - attacker becomes inactive
20   [SENSOR_AT_R] active=true -> (active'=false)&(tamper'=false);
21
22 endmodule
```

7.3 Supporting software architecture design within the SDLC

The proposed approach offers valuable capabilities for system architects and designers in the software domain. Firstly, it facilitates the assessment of architecture breaches during system development. It is widely recognized that addressing software errors and defects, especially those related to system architecture, in later stages of development incurs significant costs. By adopting the proposed approach, system architects and designers can proactively identify and rectify issues before any code is written, thereby mitigating potential problems.

Secondly, this approach enhances their understanding of security requirements. Effective software architecture modeling approaches enable critical evaluation by questioning the validity of each requirement. As a result, this comprehensive approach seamlessly integrates into various Systems Development Life Cycles (SDLCs) as a valuable supplement to the requirements specification and architectural design phases.

To illustrate its integration into the Royce iterative waterfall SDLC, the following steps are proposed:

- In the requirements specification phase, extend it by incorporating "Formalize the (New) Functional Requirement" in terms of properties.
- Expand the architecture design phase to include activities such as “Model Software Architecture” and “Analysis”. These activities aim to ensure that the software architecture model meets the desired properties of the designed system.
- Finally, if it is determined that the system design fails to satisfy desired properties, appropriate actions are taken to revisit both system requirements and/or its design.

By iterating over these phases within SDLCs, the architectural design of the system can be continuously refined and enhanced based on feedback and insights gained during each iteration. This approach empowers system architects and designers to deliver higher-quality software systems with improved security and reduced costs.

7.4 Discussion

We have demonstrated the effectiveness of formal methods, specifically PRISM-games model checking, in modeling and verifying communication modes handled by Edge Bridges. The models were developed based on the semantics of components and connectors. Functional and non-functional properties are expressed using rPATL and verified using the PRISM-games model checker. These properties included ensuring that water flow payloads sent from the cloud to the actuators are sensitive to attacks that modify sensor values and potentially disable forecasting. In our previous publication [25], we provided detailed information about the implementation of forecasting through classes represented as states. The existing research [73], proposes an ensemble-based framework for botnet detection in IoT networks using machine learning. Our method incorporates user interaction within a closed-loop system to mitigate attacks and optimize energy consumption through the application of *rigorous mathematical formalism*.

Section 5 formalizes the definitions of the components-port-connector formalism in PRISM modeling language (CSG) and elucidates the mechanism of synchronization between modules through non-player modules, which we have defined as connectors. This approach ensures an accurate equivalence between the initial component and connector model, as well as their composition on ports with labeled commands using actions in PRISM. Furthermore, it explains how component and connector ports are represented in the model, catering not only to game theory specialists but also to professionals in the software architecture domain.

8 Conclusion

In this paper, we have successfully implemented the software architecture concepts using the component-port-connector paradigm in the PRISM language. The semantics of the captured model have been implemented in CSG, where components are treated as players. We have demonstrated the impact of security concepts such as tampering on the overall architecture and its relationship to energy consumption.

The application of our approach has been specifically tailored to the Edge as a proxy between southbound and northbound bridges, with support for data serialization from low-cost protocols to more expensive ones. To evaluate the feasibility of the Edge-device-cloud architecture under external attacks, we have expressed security properties using the formalism supported by the PRISM-games model checker in rPATL. The experiments have been applied to an industrial use case addressing security and energy concerns.

For future work, we have identified three main objectives: First, Implement a robust transformation engine to map CPC language to the PRISM language, second Optimize the verification process by incorporating model abstraction techniques, and finally Explore the potential complementarity of our approach with proof assistant techniques.

A Supplementary Material

In this section, we present the key definition related to the theory behind CSGs [74, 52, 21], aiming to provide readers with a comprehensive understanding to facilitate their grasp of the knowledge required for model checking of components and connectors. In addition, our prior publications are based on the following materials [75, 76, 77, 48, 78]

A.1 Probabilistic automata

Probabilistic Automata (PA) [74] is a modeling formalism that exhibits probabilistic and nondeterministic features. Definition 4 formally illustrates a PA where $Dist(S)$ denotes the set of convex distributions over the set of states S and μ is a distribution in $Dist(S)$ that assigns a probability $\mu(s_i) = \lambda_i$ to the state $s_i \in S$.

Definition 4 (Probabilistic Automata [74]) A Probabilistic automata is a tuple $M = \langle s_0, S, \Sigma, AP, L, \delta \rangle$:

- s_0 is an initial state, such that $s_0 \in S$,
- S is a set of states,
- Σ is a finite set of actions,
- $L : S \to 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP), and
- $Pr : S \times \Sigma \to Dist(S)$ is a probabilistic transition function assigning for each $s \in S$ and $\alpha \in \Sigma$ a probabilistic distribution $\mu \in Dist(S)$.

For PA's composition, this concept is modeled by the parallel composition [74, 79]. During synchronization, each PA resolves its probabilistic choice independently [74, 79]. For transitions, $s_1 \xrightarrow{\alpha} \mu_1$ and $s_2 \xrightarrow{\alpha} \mu_2$ that synchronize in α then the composed state (s_1', s_2') is reached from the state (s_1, s_2) with probability ($\mu_1(s_1') \times \mu_2(s_2')$). In the no synchronization case, a PA takes a transition where the other remains in its current state with probability one.

A.2 Concurrent stochastic games

Concurrent stochastic multi-player games (CSGs) [52, 21] are built on the idea that CSG players make choices concurrently in each state and then transition simultaneously. Each player has control over one or more modules, and the actions that label commands within a player's modules must only be used by that specific player. The CSG is defined as an extension of PA as follows:

Definition 5 A concurrent stochastic game (CSG) [52] is a tuple $G = \langle s_0, S, N, A, \Delta, \delta, AP, L \rangle$:

- s_0 is an initial state, such that $s_0 \in S$,
- S is a set of states,
- $N = \{1, ..., n\}$ is a finite set of players,
- $A = \Sigma_1 \times \ldots \times \Sigma_n$ where Σ_i is a finite set of actions available to player $i \in N$,
- $\Delta = S \longrightarrow 2^{\cup_{i=0}^{n} \Sigma_i}$ is an action assignment function,
- $\delta : S \times A \longrightarrow Dist(S)$ is a probabilistic transition function assigning for each $s \in S$ and $(\alpha_0, \ldots, \alpha_n) \in \Sigma_i$ a probabilistic distribution $\mu \in Dist(S)$, and
- $L : S \longrightarrow 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP).

In each state s, every player i , where i belongs to the set of all players N , chooses an action from the set of available actions for that player [52, 21]. A path π of a CSG G is a sequence $\pi = s_0 \xrightarrow{\alpha_i} s_1$ [52, 21] where $s_i \in S$, $\alpha_i = (a_j^1, ..., a_j^n) \in A$, $\alpha_i^j \in Action_i(s_i)$ for $i \in N$ and $\delta(s_j, \alpha_j)(s_{j+1}) > 0$ for all $j > 0$.

CSGs are augmented with reward structures [52] as $r_A : S \times A \longrightarrow \mathbb{R}$ which assigns each state and action tuple to a real value that is accumulated when the action tuple is selected and $r_s : S \longrightarrow \mathbb{R}$ is a state reward function which assigns each state to a real value that is accumulated when the state is reached.

Properties are expressed in rPATL [53] (reward Probabilistic Alternating Temporal Logic). The property grammar is based on CTL [80] extended with coalition operator $\langle\langle C \rangle\rangle$ of ATL [81] and probabilistic operator $\mathtt{P}$ of PCTL [82]. For example, The following system property, stated in natural language: "*Players 1 and 2 can cooperate using a strategy to guarantee the probability of a robot collision within 200 steps is less than 0.003*" is expressed in rPATL as: $\langle\langle 1,2 \rangle\rangle \mathtt{P}_{<0.003}[\mathtt{F}^{\leq 200} \mathtt{collision}]$. Here, " `collision` " is the label that refers to the system states. Concerning rewards structure, the property stated in natural language: "*What is the maximum commutative reward r within 200 steps to reach* " `collision avoidance` " *for both Players 1 and 2?*" is expressed in rPATL as $\langle\langle 1,2 \rangle\rangle \mathtt{R}_{max=?}[\mathtt{C}^{\leq 200}]$

Example A.1

Consider the CSG shown in Fig. 19 [48], which corresponds to two players repeatedly performing a scheduled push and pull operation. Transitions are labeled with actions where $A = (r_1 r_2), (w_1 w_2), (w_2 r_1), (r_2 w_2), (reset_1 reset_2)$ (the symbol r refers to *pull* and the symbol w refers to *push*). The CSG starts in state " s_0 ", and states " s_1 ", " s_2 ", and " s_3 are labeled with atomic propositions corresponding to a player winning. Each player is involved through push and pull operations."

When Player 1 participates in the game and completes the pushing task, the property can be expressed as: $\langle\langle 1,2 \rangle\rangle \mathtt{P}_{>0.99} =?[\mathtt{F\ win} = 1]$.

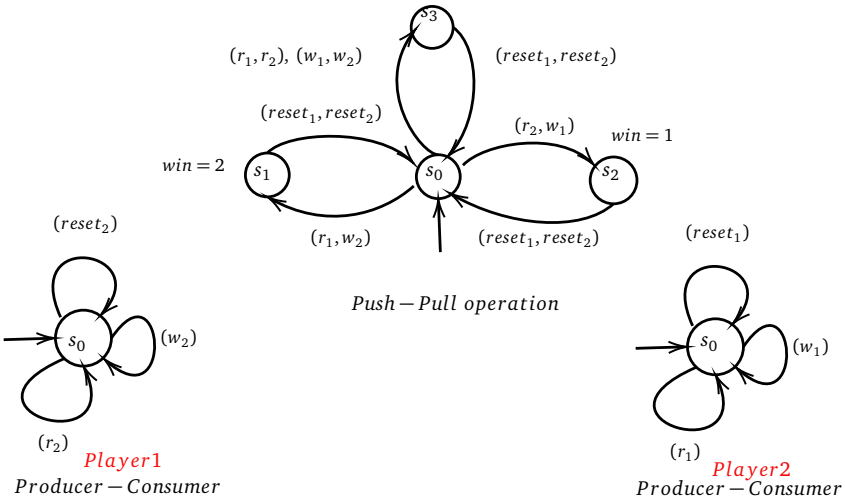

Fig. 19: Push and Pull Game Model in CSG[48].

A.3 The PRISM language

We rely on the CSG formalism [52] to express the game in PRISM language [46] to precisely capture and represent the semantics of components and connectors. The PRISM model is composed of a set of modules that can synchronize. Each module has a well-defined set of variables and commands [83] referred to as $\mathscr{D} = \langle \vartheta, \theta \rangle$. We refer to $[\![\mathscr{D}]\!]$ as a finite set of current commands. The values assigned to the variables represent the current state of the module. Each module's behavior is formally specified using a set of commands (i.e., transitions). A command is expressed as: $[a_j, \ldots, a_m]g \rightarrow \lambda_1 : u_1 + \ldots + \lambda_n : u_n$, which means, for actions " a " that are enabled only when a guard condition " g " is satisfied. When enabled, an update " u_i " occurs with a probability of " λ_i ". A guard is a Boolean expression formed by evaluating variables and applying propositional logic operators [78, 79, 75]. The update denoted by " u_i " represents an evaluation of variables, formulated as a conjunction of assignments: $v'_i = val_i + \ldots + v'_n = val_n$ where " v_i " are local variables and val_i are values evaluated via " θ " such that $\theta : \vartheta \rightarrow \mathbb{D}$ associates each variable in ϑ with a value in $\mathbb{D}$ such as $\mathbb{D} = \mathbb{N} \cup \{true, false\}$.

Declarations

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Competing interests

The authors declare no conflict of interest.

Acknowledgement

The research leading to the presented results was conducted within the research profile of s**A**fety and se**C**ur**I**ty as**S**urrance for critical **IoT** systems (ACIS-IOT), supported by the Centre National de la Recherche Scientifique (CNRS). The authors cordially thank Kentyou Company for their valuable technical support.

Availability of data and materials

The resources related to the article are available on the website `https://acis-iot.github.io/soco24.html` .

Informed Consent

Informed consent was obtained from all individual participants included in the study.

References

1. Muhammad Hafiz Hasan, Mohd Hafeez Osman, Novia Indriaty Admodisastro, and Muhamad Sufri Muhammad. Legacy systems to cloud migration: A review from the architectural perspective. *Journal of Systems and Software*, 202:111702, 2023. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2023.111702. URL https://www.sciencedirect.com/science/article/pii/S0164121223000973.
2. Vinicius F. Vidal, Leonardo M. Honório, Milena F. Pinto, Mario A.R. Dantas, Maria Júlia Aguiar, and Miriam Capretz. An edge–fog architecture for distributed 3d reconstruction. *Future Generation Computer Systems*, 135:146–158, 2022. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2022.04.015. URL https://www.sciencedirect.com/science/article/pii/S0167739X22001388.
3. Abhishek Hazra, Pradeep Rana, Mainak Adhikari, and Tarachand Amgoth. Fog computing for next-generation internet of things: Fundamental, state-of-the-art and research challenges. *Computer Science Review*, 48:100549, 2023. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2023.100549. URL https://www.sciencedirect.com/science/article/pii/S1574013723000163.
4. Claudio Cicconetti, Marco Conti, and Andrea Passarella. Uncoordinated access to serverless computing in mec systems for iot. *Computer Networks*, 172:107184, 2020. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2020.107184. URL https://www.sciencedirect.com/science/article/pii/S1389128619313684.
5. Abdelhakim Baouya, Salim Chehida, Saddek Bensalem, and Marius Bozga. Fog computing and blockchain for massive iot deployment. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2020. doi: 10.1109/MECO49872.2020.9134098.
6. Leonan T. Oliveira, Luiz F. Bittencourt, Thiago A.L. Genez, Eyal de Lara, and Maycon L.M. Peixoto. Enhancing modular application placement in a hierarchical fog computing: A latency and communication cost-sensitive approach. *Computer Communications*, 216:95–111, 2024. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2024.01.002. URL https://www.sciencedirect.com/science/article/pii/S0140366424000021.
7. Haixing Wu, Jingwei Geng, Xiaojun Bai, and Shunfu Jin. Deep reinforcement learning-based online task offloading in mobile edge computing networks. *Information Sciences*, 654:119849, 2024. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2023.119849. URL https://www.sciencedirect.com/science/article/pii/S0020025523014342.
8. Gang Zhao, Feng Zhang, Le Yu, Hongyang Zhang, Qin Qiu, and Sijia Xu. Collaborative 5g multiaccess computing security: Threats, protection requirements and scenarios. In *2021 ITU Kaleidoscope: Connecting Physical and Virtual Worlds (ITU K)*, pages 1–8, 2021. doi: 10.23919/ITUK53220.2021.9662088.
9. Majjari Sudhakar and Koteswara Rao Anne. Optimizing data processing for edge-enabled iot devices using deep learning based heterogeneous data clustering approach. *Measurement: Sensors*, 31:101013, 2024. ISSN 2665-9174. doi: https://doi.org/10.1016/j.measen.2023.101013. URL https://www.sciencedirect.com/science/article/pii/S2665917423003495.
10. Rohit Kumar and Neha Agrawal. Analysis of multi-dimensional industrial iot (iiot) data in edge–fog–cloud based architectural frameworks : A survey on current state and research challenges. *Journal of Industrial Information Integration*, 35:100504, 2023. ISSN 2452-414X. doi: https://doi.org/10.1016/j.jii.2023.100504. URL https://www.sciencedirect.com/science/article/pii/S2452414X23000778.
11. Javad Dogani, Reza Namvar, and Farshad Khunjush. Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey. *Computer Communications*, 209:120–150, 2023. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2023.06.010. URL https://www.sciencedirect.com/science/article/pii/S0140366423002086.
12. Mobasshir Mahbub and Raed M. Shubair. Contemporary advances in multi-access edge computing: A survey of fundamentals, architecture, technologies, deployment cases, security, challenges, and directions. *Journal of Network and Computer Applications*, 219:103726, 2023. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2023.103726.
13. Piroska Haller, Béla Genge, and Adrian-Vasile Duka. *Engineering Edge Security in Industrial Control Systems*, pages 185–200. Springer International Publishing, Cham, 2019. ISBN 978-3-030-00024-0. doi: 10.1007/978-3-030-00024-0_10.
14. Abdelhakim Baouya, Samir Ouchani, and Saddek Bensalem. Formal modelling and security analysis of interoperable systems. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Yinglin Wang, editors, *Advances and Trends in Artificial Intelligence. Theory and Practices in Artificial Intelligence*, pages 555–567. Springer International Publishing, Cham, 2022. ISBN 978-3-031-08530-7.
15. Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Axel Legay, and Saddek Bensalem. Mitigating security risks through attack strategies exploration. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification*, volume 11245, pages 392–413. Springer International Publishing. ISBN 978-3-030-03420-7 978-3-030-03421-4. doi: 10.1007/978-3-030-03421-4_25. URL http://link.springer.com/10.1007/978-3-030-03421-4_25. Series Title: Lecture Notes in Computer Science.
16. Quentin Rouland, Brahim Hamid, and Jason Jaskolka. Reusable formal models for threat specification, detection, and treatment. In Sihem Ben Sassi, Stéphane Ducasse, and Hafedh Mili, editors, *Reuse in Emerging Software Engineering Practices*, pages 52–68, Cham, 2020. Springer International Publishing. ISBN 978-3-030-64694-3.
17. Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. Stride-based threat modeling for cyber-physical systems. In *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6, 2017. doi: 10.1109/ISGTEurope.2017.8260283.
18. Ann Yi Wong, Eyasu Getahun Chekole, Martín Ochoa, and Jianying Zhou. On the security of containers: Threat modeling, attack analysis, and mitigation strategies. *Computers & Security*, 128:103140, 2023. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2023.103140. URL https://www.sciencedirect.com/science/article/pii/S0167404823000500.

19. L. Ou, Z. Qin, H. Yin, and K. Li. Chapter 12 - security and privacy in big data. In Rajkumar Buyya, Rodrigo N. Calheiros, and Amir Vahid Dastjerdi, editors, *Big Data*, pages 285–308. Morgan Kaufmann, 2016. ISBN 978-0-12-805394-2. doi: https://doi.org/10.1016/B978-0-12-805394-2.00012-X. URL https://www.sciencedirect.com/science/article/pii/B978012805394200012X.
20. Microsoft, the STRIDE Threat Model. Microsoft corporation, 2009. URL https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN.
21. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Prism-games 3.0: Stochastic game verification with concurrency, equilibria and time. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 475–487, Cham, 2020. Springer International Publishing.
22. Jinxuan Li, Qiongfang Liang, Dongxu Li, Yuan Liu, and Yang Zhang. Game-based task offloading in cloud-edge-end network. In *2023 8th International Conference on Computer and Communication Systems (ICCCS)*, pages 549–553, 2023. doi: 10.1109/ICCCS57501.2023.10151121.
23. Jing Jiang, Peizhe Xin, Yudong Wang, Lirong Liu, Yuhao Chai, Yong Zhang, and Sige Liu. Computing resource allocation in mobile edge networks based on game theory. In *2021 IEEE 4th International Conference on Electronics and Communication Engineering (ICECE)*, pages 179–183, 2021. doi: 10.1109/ICECE54449.2021.9674451.
24. Quentin Rouland, Brahim Hamid, and Jason Jaskolka. Formal specification and verification of reusable communication models for distributed systems architecture. *Future Generation Computer Systems*, 108:178–197, 2020. ISSN 0167-739X.
25. Abdelhakim Baouya, Salim Chehida, Samir Ouchani, Saddek Bensalem, and Marius Bozga. Generation and verification of learned stochastic automata using k-NN and statistical model checking. 52(8):8874–8894, . ISSN 0924-669X, 1573-7497. doi: 10.1007/s10489-021-02884-4. URL https://link.springer.com/10.1007/s10489-021-02884-4.
26. Xiao Zhang, Xiaoming Chen, Yuxiong He, Youhuai Wang, Yong Cai, and Bo Li. Neural network-based ddos detection on edge computing architecture. In *2022 4th International Conference on Applied Machine Learning (ICAML)*, pages 1–4, 2022. doi: 10.1109/ICAML57167.2022.00067.
27. Dapeng Wu, Junjie Yan, Honggang Wang, and Ruyan Wang. Multiattack intrusion detection algorithm for edge-assisted internet of things. In *2019 IEEE International Conference on Industrial Internet (ICII)*, pages 210–218, 2019. doi: 10.1109/ICII.2019.00046.
28. Yixuan Wu, Laisen Nie, Shupeng Wang, Zhaolong Ning, and Shengtao Li. Intelligent intrusion detection for internet of things security: A deep convolutional generative adversarial network-enabled approach. *IEEE Internet of Things Journal*, 10(4):3094–3106, 2023. doi: 10.1109/JIOT.2021.3112159.
29. Yan Huang and Weimeng Zhang. Research on the methods of data mining based on the edge computing for the iot. In *2023 IEEE International Conference on Integrated Circuits and Communication Systems (ICICACS)*, pages 1–6, 2023. doi: 10.1109/ICICACS57338.2023.10099991.
30. Nitish A, Hanumanthappa J, Shiva Prakash S. P, and Kirill Krinkin. On-device context-aware misuse detection framework for heterogeneous iot edge. *Applied Intelligence*, 53(12):14792–14818, Jun 2023. ISSN 1573-7497. doi: 10.1007/s10489-022-04039-5. URL https://doi.org/10.1007/s10489-022-04039-5.
31. Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835–846, 2011. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2011.01.004. URL https://www.sciencedirect.com/science/article/pii/S0164121211000069.
32. Georgia Giannopoulou, Peter Poplavko, Dario Socci, Pengcheng Huang, Nikolay Stoimenov, Paraskevas Bourgos, Lothar Thiele, Marius Bozga, Saddek Bensalem, Sylvain Girbal, Madeleine Faugere, Romain Soulat, and Benoît Dupont de Dinechin. Dol-bip-critical: a tool chain for rigorous design and implementation of mixed-criticality multi-core systems. *Design Automation for Embedded Systems*, 22(1):141–181, Jun 2018. ISSN 1572-8080. doi: 10.1007/s10617-018-9206-3. URL https://doi.org/10.1007/s10617-018-9206-3.
33. Hui Song, Rustem Dautov, Nicolas Ferry, Arnor Solberg, and Franck Fleurey. Model-based fleet deployment in the iot–edge–cloud continuum. *Software and Systems Modeling*, 21(5):1931–1956, Oct 2022. ISSN 1619-1374. doi: 10.1007/s10270-022-01006-z. URL https://doi.org/10.1007/s10270-022-01006-z.
34. Riya Kakkar, Rajesh Gupta, Smita Agrawal, Sudeep Tanwar, and Emil Pricop. Game-based resource allocation for secure uav communication in wireless networks. In *2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6, 2023. doi: 10.1109/ECAI58194.2023.10194073.
35. Zina Chkirbene, Ridha Hamila, and Aiman Erbad. Secure wireless sensor networks for anti-jamming strategy based on game theory. In *2023 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1101–1106, 2023. doi: 10.1109/IWCMC58020.2023.10182764.
36. Samira Chouikhi, Moez Esseghir, and Leila Merghem-Boulahia. Computation offloading for industrial internet of things: A cooperative approach. In *2023 International Wireless Communications and Mobile Computing (IWCMC)*, pages 626–631, 2023. doi: 10.1109/IWCMC58020.2023.10183071.
37. Bing Zhao, Kexin Zhang, Maciej Ogorzałek, Qing Gao, and Jinhu Lü. Security framework for cloud control systems against false data injection attacks. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2023. doi: 10.1109/ISCAS46773.2023.10182125.
38. Kim Hammar and Rolf Stadler. Learning near-optimal intrusion responses against dynamic attackers. *IEEE Transactions on Network and Service Management*, pages 1–1, 2023. doi: 10.1109/TNSM.2023.3293413.
39. Mattia Borgo, Bruno Principe, Lorenzo Spina, Laura Crosara, Leonardo Badia, and Elvina Gindullina. Attack strategies among prosumers in smart grids: A game-theoretic approach. In *2023 11th International Conference on Smart Grid (icSmartGrid)*, pages 01–06, 2023. doi: 10.1109/icSmartGrid58556.2023.10170768.
40. Stylianos Basagiannis, Panagiotis Katsaros, Andrew Pombortsis, and Nikolaos Alexiou. Probabilistic model checking for the quantification of dos security threats. *Computers & Security*, 28(6):450–465, 2009. ISSN 0167-4048.
41. Saif U.R. Malik, Adeel Anjum, Syed Atif Moqurrab, and Gautam Srivastava. Towards enhanced threat modelling and analysis using a markov decision process. *Computer Communications*, 194:282–291, 2022. ISSN 0140-3664.

42. Zheng Fang, Hao Fu, Tianbo Gu, Zhiyun Qian, Trent Jaeger, Pengfei Hu, and Prasant Mohapatra. A model checking-based security analysis framework for iot systems. *High-Confidence Computing*, 1(1):100004, 2021. ISSN 2667-2952.
43. Samir Ouchani, Otmane Ait Mohamed, and Mourad Debbabi. A security risk assessment framework for SysML activity diagrams. In *2013 IEEE 7th International Conference on Software Security and Reliability*, pages 227–236. IEEE. ISBN 978-1-4799-0406-8 978-0-7695-5021-3.
44. Simon Bliudze and Joseph Sifakis. The algebra of connectors—structuring interaction in BIP. 57(10):1315–1330. ISSN 0018-9340.
45. U.S. Department of Homeland Security. Common attack pattern enumeration and classification, November 2002. URL http://capec.mitre.org.
46. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
47. Sherman Kent. Ssherman kent and the profession of intelligence analysis, center for the study of intelligence, central intelligence agency, November 2002 p.55. URL https://www.cia.gov/library/kent-center-occasional-papers/vol1no5.htm.
48. Abdelhakim Baouya, Brahim Hamid, Levent Gürgen, and Saddek Bensalem. Rigorous security analysis of rabbitmq broker with concurrent stochastic games. *Internet of Things*, 26:101161, 2024. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2024.101161.
49. Gerard J. Holzmann. *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004. ISBN 978-0-321-22862-8.
50. Quentin Rouland, Brahim Hamid, and Jason Jaskolka. Specification, detection, and treatment of stride threats for software components: Modeling, formal methods, and tool support. *Journal of Systems Architecture*, 117:102073, 2021. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2021.102073. URL https://www.sciencedirect.com/science/article/pii/S1383762121000631.
51. I. Crnkovic, Michel R. V. Chaudron, and S. Larsson. Component-Based Development Process and Component Lifecycle. In *Proceedings of the International Conference on Software Engineering Advances (ICSEA)*, page 44. IEEE Computer Society, 2006.
52. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Automatic verification of concurrent stochastic systems. 58(1). ISSN 0925-9856, 1572-8102.
53. Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214, pages 315–330. Springer Berlin Heidelberg. ISBN 978-3-642-28755-8 978-3-642-28756-5.
54. Brain-iot, 2020. URL https://www.brain-iot.eu/.
55. The PRISM Model Checker Team. Concurrent stochastic games (csgs) - prism-games. https://www.prismmodelchecker.org/games/modelling.php#csgs.
56. Abdelhakim Baouya. Paper Artefacts Sources. https://acis-iot.github.io/soco24.html.
57. Frank Berkers, Marc Roelands, Freek Bomhof, Thomas Bachet, Martin van Rijn, and Wietske Koers. Constructing a multi-sided business model for a smart horizontal iot service platform. pages 126–132, 10 2013.
58. Ivan Farris, Roberto Girau, Leonardo Militano, Michele Nitti, Luigi Atzori, Antonio Iera, and Giacomo Morabito. Social virtual objects in the edge cloud. *IEEE Cloud Computing*, 2(6):20–28, 2015.
59. Roberto Morabito, Riccardo Petrolo, Valeria Loscrì, and Nathalie Mitton. Legiot: A lightweight edge gateway for the internet of things. *Future Generation Computer Systems*, 81:1 – 15, 2018. ISSN 0167-739X.
60. Bigclout, 2016. URL https://bigclout.eu/.
61. Wise iot, 2018. URL https://www.wise-iot.eu/.
62. Iof2020, 2020. URL https://www.iof2020.eu/.
63. Activage project, 2020. URL https://activageproject.eu/.
64. Abdelhakim Baouya, Salim Chehida, Saddek Bensalem, Levent Gürgen, Richard Nicholson, Miquel Cantero, Mario Diaznava, and Enrico Ferrera. Deploying warehouse robots with confidence: the brain-iot framework's functional assurance. *The Journal of Supercomputing*, 80(1):1206–1237, Jan 2024. ISSN 1573-0484. doi: 10.1007/s11227-023-05483-x. URL https://doi.org/10.1007/s11227-023-05483-x.
65. Eclipse. Sensinact/gateway architecture. https://wiki.eclipse.org/SensiNact/Gateway_Architecture, 2024.
66. Adam Shostack. *Threat Modeling: Designing for Security*. Wiley, 2014. ISBN 9781118809990.
67. Brahim Hamid and Donatus Weber. Engineering secure systems: Models, patterns and empirical validation. *Computers & Security*, 77:315–348, 2018. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2018.03.016. URL https://www.sciencedirect.com/science/article/pii/S0167404818303043.
68. PRISM Development Team (eds.). *PRISM Manual*. URL https://www.prismmodelchecker.org/manual/ConfiguringPRISM/ComputationEngines. Accessed: March 27, 2024.
69. Quanyan Zhu and Tamer Başar. Dynamic policy-based ids configuration. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 8600–8605, 2009. doi: 10.1109/CDC.2009.5399894.
70. Kamar Kesrouani, Houssam Kanso, and Adel Noureddine. A preliminary study of the energy impact of software in raspberry pi devices. In *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 231–234, 2020. doi: 10.1109/WETICE49692.2020.00052.
71. RabbitMQ - amqp 0-9-1, 2019. URL https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf.
72. Zubaida Rehman, Iqbal Gondal, Mengmeng Ge, Hai Dong, Mark Gregory, and Zahir Tari. Proactive defense mechanism: Enhancing iot security through diversity-based moving target defense and cyber deception. *Computers & Security*, 139:103685, 2024. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2023.103685. URL https://www.sciencedirect.com/science/article/pii/S0167404823005953.

73. Jincheng Zhou, Tao Hai, Dayang Norhayati Abang Jawawi, Dan Wang, Kuruva Lakshmanna, Praveen Kumar Reddy Maddikunta, and Mavellous Iwendi. A lightweight energy consumption ensemble-based botnet detection model for iot/6g networks. *Sustainable Energy Technologies and Assessments*, 60:103454, 2023. ISSN 2213-1388. doi: https://doi.org/10.1016/j.seta.2023.103454. URL https://www.sciencedirect.com/science/article/pii/S2213138823004472.
74. V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.
75. Abdelhakim Baouya, Djamal Bennouar, Otmane Ait Mohamed, and Samir Ouchani. A quantitative verification framework of sysml activity diagrams under time constraints. *Expert Systems with Applications*, 42(21):7493–7510, 2015. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2015.05.049. URL https://www.sciencedirect.com/science/article/pii/S0957417415003851.
76. Abdelhakim Baouya, Djamal Bennouar, Otmane Ait Mohamed, and Samir Ouchani. A quantitative verification framework of SysML activity diagrams under time constraints. 42(21):7493–7510, . ISSN 09574174.
77. Abdelhakim Baouya, Otmane Ait Mohamed, and Samir Ouchani. Toward a context-driven deployment optimization for embedded systems: a product line approach. *The Journal of Supercomputing*, 79(2):2180–2211, feb 2023. ISSN 1573-0484. doi: 10.1007/s11227-022-04741-8. URL https://doi.org/10.1007/s11227-022-04741-8.
78. Abdelhakim Baouya, Otmane Ait Mohamed, Samir Ouchani, and Djamal Bennouar. Reliability-driven automotive software deployment based on a parametrizable probabilistic model checking. *Expert Systems with Applications*, 174:114572, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.114572.
79. Samir Ouchani, Otmane Aït Mohamed, and Mourad Debbabi. A formal verification framework for sysml activity diagrams. *Expert Systems with Applications*, 41(6):2713–2728, 2014. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2013.10.064. URL https://www.sciencedirect.com/science/article/pii/S0957417413008968.
80. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press. ISBN 978-0-262-02649-9. OCLC: ocn171152628.
81. Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, sep 2002. ISSN 0004-5411.
82. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. 6(5):512–535. ISSN 0934-5043, 1433-299X.
83. Marta Kwiatkowska, Gethin Norman, and David Parker. Quantitative analysis with the probabilistic model checker prism. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, 2006. ISSN 1571-0661. Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).

On the Sound Applicability of Model Checking and Simulation for Verifying Security Properties of AMQP Direct Message Exchanges: A Roadmap within the Model-Driven Engineering Flow

Abdelhakim Baouya[a,*], Brahim Hamid[a], Saddek Bensalem[b]

[a]*University of Toulouse, IRIT, France*
[b]*Université Grenoble Alpes, VERIMAG, Grenoble, France*

Abstract

Modern distributed systems architectures encompass diverse computational logic and communication protocols. However, the inherent security vulnerabilities in these systems pose significant risks at the different layers. In this work, we focus on reducing the complexity of security analysis at the middleware communication layer through modeling and analyzing the well-known AMQP direct exchange messages. We propose an approach that leverages Probabilistic Automata (PA) and a simulation platform to model the behavior of the Messages Exchange while considering potential data corruption attacks. We implement the PA model using PRISM for automated analysis and develop a simulation model using OMNeT++. Collected OMNeT++ simulation traces are then used to learn Probabilistic Decision Tree (PDT) to build a more abstract PA model. The soundness of these two PA models is compared, mainly against the incorporation of the security specification for analysis. To validate our approach, we investigate the impact of threats on payload routing and message loss for water dam infrastructure and identify gaps in security using verification and simulation. Building a more tractable probabilistic model by simulation and learning has improved scalability, as evidenced by comparative verification times.

Keywords: Software architecture, Clock drifts, Decision Trees, Formal methods.

Contents

*Corresponding author at :University of Toulouse, IRIT, France
Email addresses: abdelhakim.baouya@irit.fr, abaouya@acm.org (Abdelhakim Baouya), brahim.hamid@irit.fr (Brahim Hamid), saddek.bensalem@univ-grenoble-alpes.fr (Saddek Bensalem)

1. Introduction

The Internet of Things (IoT) has transformed our interaction with technology and our environment, offering a plethora of benefits and opportunities. However, alongside these advantages come substantial security challenges [1]. Insecure network connections within IoT systems present risks such as data interception, unauthorized access, and tampering [2]. Consequently, it is crucial to identify and mitigate these security concerns right from the initial stages of development [3]. This highlights the importance of establishing precise semantic definitions for attacks during the architecture design phase. By doing so, we can enhance the security posture of IoT systems and ensure their resilience in the face of evolving cyber threats.

IoT devices rely on communication middleware to establish seamless connectivity and exchange data with the central system that utilizes application-level protocols. Two commonly used device-side protocols: MQTT [4] and CoAP [5]. However, these protocols are not deemed reliable regarding data management at the Edge. They lack trustworthiness in effectively handling incoming sensed data at varying frequencies compared to RabbitMQ[1]. RabbitMQ[2] as a core-side broker that stands out as a message queue middleware that implements asynchronous message communication of the AMQP and is widely adopted by reputable vendors such as JPMorgan, NASA, Google, IBM, VMware, and others.

Understanding the behavior of communication protocols independently of the platform, especially those in the IoT domain, requires modeling the system at different levels of abstraction [6]. This allows us to pinpoint the details relevant to the intent modeling objectives. Model-driven Engineering (MDE) [7, 8, 9] offers a systematic approach to addressing functional and security concerns in IoT protocols from various perspectives, including the modeling level and artifact level [10]. The modeling level focuses on the architectural specifications independently of the execution systems. It can be portrayed graphically or textually. The artifact level deals with the generated code produced by the automation procedure. This code is created by mapping the models to their corresponding code equivalents, ensuring compatibility with the execution system. For instance, the work in [11] establishes a workflow within the Model-Driven Engineering (MDE) paradigm. Artifacts are modeled in BIP for robot orchestration, verified using statistical model checking, and simulated after code generation.

Simulation is a powerful tool for software designers to uncover defects, errors, and implementation flaws. However, defining a comprehensive set of scenarios and patterns that fully cover the intended system properties can be challenging. Formal methods [12, 13], on the other hand, offer valuable insights into functional and security properties, especially those related to resource-constrained protocols [14, 15]. Integrating both approaches seamlessly into the MDE design flow to provide quality indicators for informed decision-making by designers remains a complex challenge that can be translated as a research question: *Is there formal proof that the artifact level is faithful to the model's specifications?*

1.1. Contribution

Our work explores how to bridge the gap between formal verification techniques and the expertise of simulation practitioners. This research paper investigated the AMQP direct message exchange system [16] as the target system for analysis within the PRISM and OMNeT++ simulation frameworks. The PRISM model is then verified against security properties expressed in temporal logic, while the OMNeT++ model is validated for the same properties. Furthermore, the models incorporated an attack model at the design stage to create a system suitable for both verification and simulation. This combined model served as a unified artifact for both purposes. In a unique approach, the OMNeT++ model was simulated to generate execution traces. These traces were then used to learn and construct a formal model in PRISM. The resulting PRISM model is also verified against PCTL properties.

Two research questions shall be adressed to ensure that the artifact level is faithful to the model's specifications: (1) Can the formal model constructed from simulation and learning be compared to the original design-stage model? (2) Will the learned model address scalability issues faced by the original PRISM model at the design stage?

[1] https://www.educba.com/rabbitmq-vs-mqtt/
[2] https://www.rabbitmq.com/

By comparatively analyzing models built with different formalisms, we aim to provide valuable insights for designers. These insights will focus on effective techniques for verifying temporal logic properties and ensuring that models built at the artifact stage accurately reflect those at the modeling stage.

1.2. Outline

The subsequent sections of this paper are organized as follows: In Section 2, we present the necessary preliminaries related to the modeling in PRISM, OMNeT++, and learning using the Probabilistic Decision Tree. Section 3 provides an in-depth explanation of the approach upon which our paper relies. Subsequently, in Section 4, we formally define the behavioral semantics of the AMQP communication behavior. In Section 5, we capture the semantic behavior of the AMQP in PRISM and OMNeT++. In Section 6, we conduct an experiment within the context of IoT systems to illustrate the practical application of our approach regarding scalability. In Section 7, we summarize the existing related works to provide a broader context for our research. Finally, in Section 8, we conclude our work and propose potential avenues for future research.

2. Theoretical Background

In this section, we will introduce the two frameworks that we utilized in our work: the PRISM language [17] for analysis through formal verification and OMNeT++ [18] for modeling a network of Cyber-Physical systems. In addition, we provide a theoretical overview of Probabilistic Decision Trees (PDT). Here, The formal theoretical foundations provide a knowledge base for researchers in academia and industry to model complex cyber-physical systems and machine learning.

2.1. Modeling with PRISM

We rely on the probabilistic automata model to express the stochastic behavior in PRISM language [17]. The PRISM model is composed of a set of modules that can synchronize. A set of variables and commands characterizes each module. The variable's valuations represent the state of the module. A set of commands is used to describe the behavior of each module (i.e., transitions). A command takes the form: $[a]g \rightarrow \lambda_1 : u_1 + \ldots + \lambda_n : u_n$, which means, for actions " a " if the guard " g " is true, then, an update " u_i " is enabled with a probability " λ_i ". A guard is a logical proposition consisting of variable evaluation and propositional logic operators. The update " u_i " is an evaluation of variables expressed as a conjunction of assignments: $v_i' = val_i + \ldots + v_n' = val_n$ where " v_i " are local variables. Each command is encapsulated within a PRISM module [19]. The set of updates of PRISM Model is denoted by $[\![\mathscr{P}_{\mathscr{M}}]\!]$ such that a command update $t \in \mathscr{C}$, where $\mathscr{P}_{\mathscr{M}} = \langle X, \mathscr{C} \rangle$, X is a set of valuations on a set of variables ϑ and $\mathscr{C}$ is the PRISM commands.

2.2. Modeling with OMNeT++

OMNeT++ [18] is a versatile C++ simulation library and framework specifically designed for building network and Cyber-Physical simulators. It offers extensibility, modularity, and a component-based approach. The modeling framework comprises two essential components: the NED topology description language and the component code written in C/C++. The NED modeling language is composed of a set of components and connections.

Definition 1. A NED component " $\mathscr{ND}$ " is a tuple $\mathscr{ND} = \langle \vartheta, \Sigma, \mathcal{E} \rangle$, where:

- ϑ is a finite set of component variables and parameters associated with the component. Also, we consider the X as valuation over ϑ and X' as its new valuation.
- Σ is a finite set of gates that can take the form of $input$ (formally ?) and $output$ (formally !). We identify by τ as internal behavior related to the OMNeT component.
- $\mathcal{E} \subseteq X \times \Sigma \times X'$ is state transition function executed triggered by the gates Σ .

While simulating with OMNeT++, the state of the component denoted by $\mathscr{ND}$ takes values from the set X , so, we establish semantics for the $\mathscr{ND}$ state based on component variables as follows:

Definition 2 ($\mathcal{ND}$ state)**.** A $\mathcal{ND}$ state is the valuation of $\mathcal{ND}$ components variables $v_0, \ldots, v_n \in \vartheta$ in the form $s = \langle x_0, \ldots, x_n \rangle$.

The interaction between $\mathcal{ND}$ components leads to a series of updates on variable values. We define a sequence of $\mathcal{ND}$ executions as follows :

Definition 3 (Sequence of $\mathcal{ND}$ execution)**.** A Sequence of $\mathcal{ND}$ executions is a continuous update of $\mathcal{ND}$ components variables of $\langle x_0, \ldots, x_n \rangle \xhookrightarrow{a} \langle x'_0, \ldots, x'_n \rangle$, where $a \in \Sigma$ and $x'_0, \ldots, x'_n$ is the update of variable $v_0, \ldots, v_n$ based on function $\mathcal{E}$. The set of updates of NED components execution is denoted by $[\![Seq_{\mathcal{ND}}]\!]$ such that for an update $t = (s, a, s'),\ t \in [\![Seq_{\mathcal{ND}}]\!]$.

2.3. Probabilistic Decision Trees

Probabilistic Decision Trees (PDTs) [20] are a decision tree model that incorporates probabilistic information into the decision-making process. Unlike traditional decision trees that make deterministic decisions based on fixed rules, PDTs assign probabilities to each possible outcome at each decision node. Considering a dataset of inference inputs features I and a set of output features Q we define PDTs as follows:

Definition 4. A PDT is a tuple $\mathcal{T} = \langle I, G, \Sigma, Rule \rangle$ such that:

- I : A set of input features.
- Q : is a set of output features.
- Σ It is a finite set of action labeling the decision tree's rules.
- $Rule$: $Const(I) \rightarrow Dist(Q)$ is a probabilistic rule expressed in the form of $g_0 \wedge \ldots \wedge g_n \xhookrightarrow{d}_{\lambda} q_0 \wedge \ldots \wedge q_n$ where $d \in \Sigma,\ g_0 \wedge \ldots \wedge g_n \in Const(I)$ is a set of constraint over the input features I , and $\lambda \in Dist(Q)$ represents the probability of occurrence of the rule in the dataset. The set of rules $r_0, \ldots, r_n$ generated by the decision trees is denoted by $[\![\mathcal{T}]\!]$ such that a rule $r_i \in [\![\mathcal{T}]\!]$.

The example in Figure 1 shows the structure of the Probabilistic Decision Tree (PDT), where each node represents a feature, except for the leaf node, which represents the output feature. In this specific case, we can interpret the following rule: `temperature >= 29` $\wedge$ `humidity >= 80%` $\rightarrow_1$ `Ventilation level= 2`

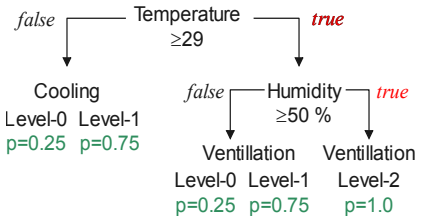

Figure 1: Example of a Probabilistic Decision Tree (PDT)

3. Approach

In this paper, we propose to implement the approach depicted in Figure 2 for modeling and learning for security analysis in the context of the AMQP protocol. The modeling phase involves the interpretation of protocol specifications from natural language into PRISM and OMNeT++ models, handled by two different actors.

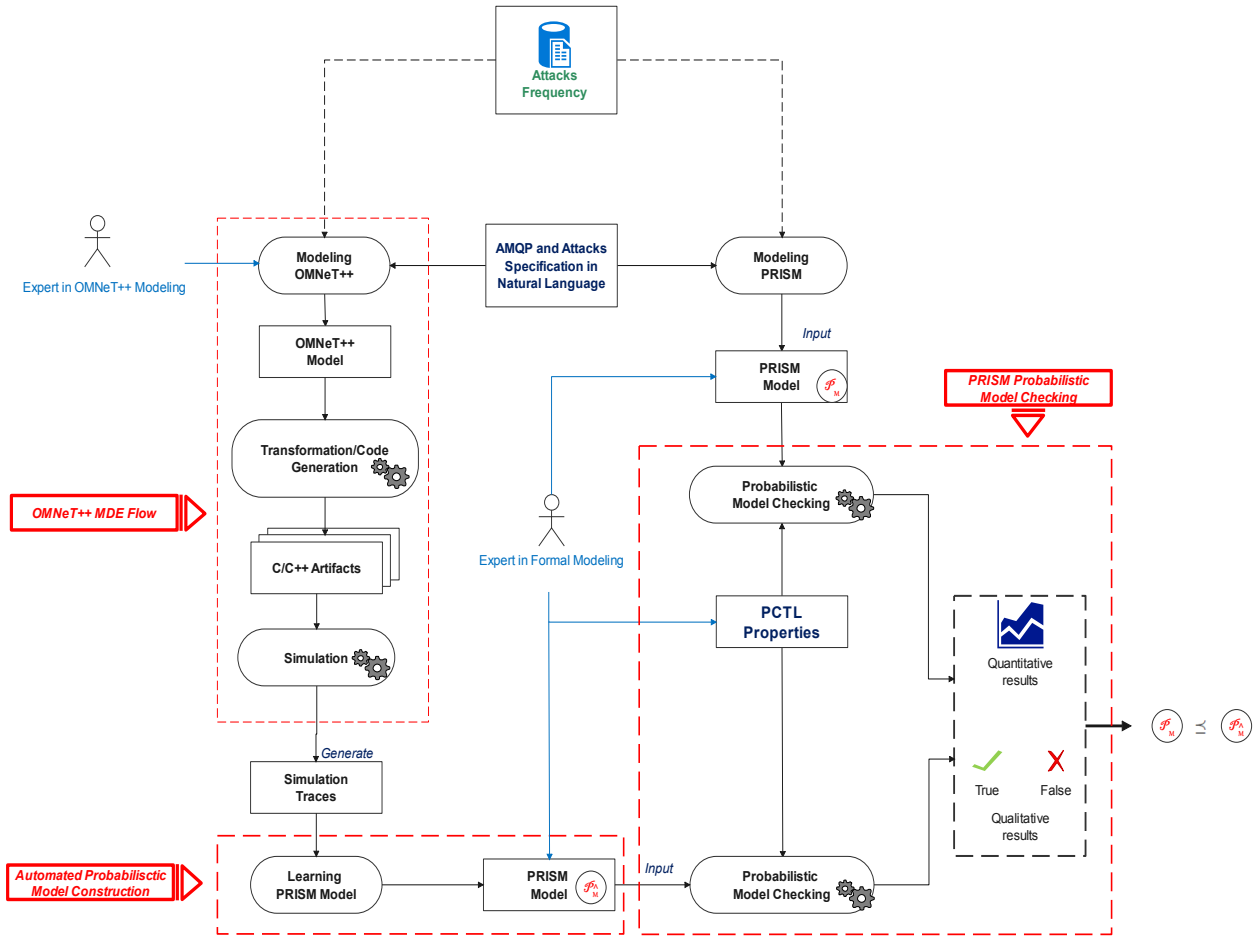

Figure 2: Modeling and Learning using PRISM, OMNeT++, and Probabilistic Decision Trees.

Modeling with PRISM. The PRISM model captures the semantic behavior (referred to as $\mathscr{P}_{\mathscr{M}}$) of the AMQP protocol, encompassing entities such as producers and consumers. However, the inherent abstractness of Model M1 poses challenges when it comes to specifying attack semantics related to security. To overcome this challenge, security aspects are modeled at the same level of abstraction, utilizing the language constructs provided by the model to incorporate well-documented attack scenarios (CAPEC-384 [21]). On the other hand, the OMNeT++ model implements the AMQP protocols with a focus on message management. The precise specification with OMNeT (close to the code level) offers more facilities to integrate security aspects while specifying attack semantics. Attack scenarios can be implemented using its semantics with the help of C/C++ constructs, while PRISM has certain limitations in this regard. Additionally, the frequencies of attacks can be gathered from various data sources and incorporated into the model. This allows for a more comprehensive and realistic representation of the system's security landscape.

Modeling with OMNeT++. At the OMNeT++ modeling level, the AMQP protocol employs a First-In-First-Out (FIFO) mechanism for enqueueing and dequeueing incoming and outgoing data. The learning function captures microstreaming traces from the C/C++ artifacts simulation and generates a corresponding PRISM model, referred to as $\mathscr{P}_{\widehat{\mathscr{M}}}$, utilizing Probabilistic Decision Trees (PDT). This crucial step is handled by an expert in the field of learning. The learning process from the simulation traces provides a pattern of how the protocol behaves in the presence of attacks. Subsequently, both the $\mathscr{P}_{\mathscr{M}}$ and $\mathscr{P}_{\widehat{\mathscr{M}}}$ models are subjected to rigorous verification against PCTL properties that express the system's security requirements. The results obtained from probabilistic model

checking for these two models are compared using correlation analysis. This comparison serves as a means to gauge the quality of the models obtained through the learning process ($\mathscr{P}_{\widehat{\mathscr{M}}}$) and the initial modeled system $\mathscr{P}_{\mathscr{M}}$). Verification times on $\mathscr{P}_{\widehat{\mathscr{M}}}$ shall demonstrate the approach's effectiveness in handling model scalability in building tractable models without human intervention.

Automation. Our approach leverages the Model-Driven Engineering (MDE) flow within OMNeT++. This allows us to model the system network and automatically generate C/C++ code suitable for simulation using the QT simulator. Subsequently, we collect and process the simulation traces to perform a learning procedure and generate a set of PPDT rules, denoted by $\mathscr{T}$.

Scenario. The approach is performed over a use case scenario from the collaborative European research project on smart water flow assessment (BrainIoT 2022 [22]). The scenario involves three sensor nodes that collect and transmit various water-related measurements, such as water level (WL), water volume (WV), and rain precipitation (RP), to a fog computing system via the Sensinact gateway [23]. This data is then used to assess water flow (WF). Our focus here is on the AMQP protocol used for communication, which is considered vulnerable to attacks within the edge infrastructure.

4. The Formal Modeling of AMQP Architecture and Attacks Realization

This section presents our modeling of the AMQP protocol for direct message exchanges. We focus on data modification attacks implementing CAPEC-384 [21] that could potentially impact the integrity of messages exchanged between communicating entities. The formal operational semantics rules rely on the theoretical representation of rule derivation in [24].

4.1. The AMQP architecture model

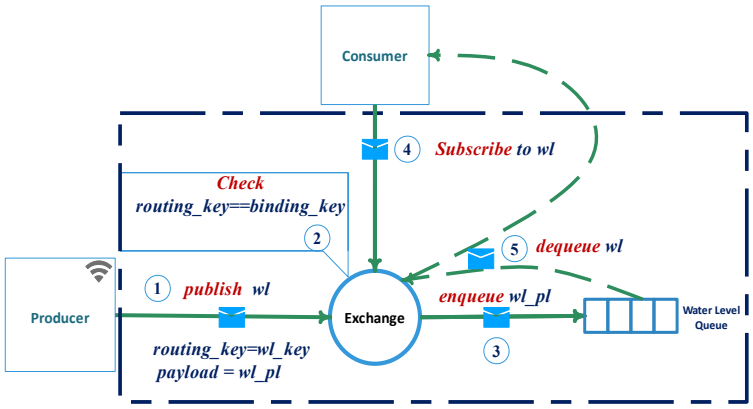

Figure 3: The AMQP Architecture Model.

As visualized in Figure 3, the exchange receives messages from the producers (1) and routes them (3) to message queues based on the specified **binding key** (2). The exchange analyzes the message properties the producer provides to extract the **routing key**, which serves as a virtual address to determine the appropriate message queue for storing (i.e., enqueuing) the message by matching the binding keys. The consumer subscribes to changes in the queue (4) and can consume (i.e., dequeuing) messages (5). Formally, the AMQP system during execution is defined as follows:

Definition 5. (AMQP). The AMQP protocol " $\mathscr{RQ}$ " is a tuple $\mathscr{RQ} = \langle \mathscr{M}, \mathscr{K}, \mathcal{A}, Map \rangle$, where:

- $\mathscr{M}$ is the message structure in $\mathscr{RQ}$,
- $\mathscr{K} = \{k_0, \ldots, k_n\}$ is a set of keys (i.e., binding or routing keys),

- $\mathcal{A}$: A set of channels is utilized by the exchange to receive incoming and outgoing messages.
- Map : $\mathscr{K} \rightarrow Q$, is a mapping function that assigns each key to a specific queue queue supported by $\mathscr{R}\mathscr{2}$.

Throughout the paper, we consider a message composed of a *routing key and a payload*. Formally, we define an AMQP message as:

Definition 6. (Message). An AMQP message " $\mathscr{M}$ " is a tuple $\mathscr{M} = \langle k, pld \rangle$, where:

- k is the routing key of the sensor message and
- pld is the payload of the sensed data

We also write $[\![\mathscr{R}\mathscr{2}]\!]$ to denote a set of messages at the front of the queues associated with each Map . So, we define $\mathscr{R}\mathscr{2}$ message status as:

Definition 7. ($\mathscr{R}\mathscr{2}$ queue status). An AMQP queue status s is the message at the front of the queue map_k noted $[\![\mathscr{R}\mathscr{2}_{\lceil map_k}]\!]$ associated with key k . Whereas, a set of operational transitions of producers and consumers are modeled as $[\![\mathscr{R}\mathscr{2}_{\lceil p}]\!]$ and $[\![\mathscr{R}\mathscr{2}_{\lceil c}]\!]$, respectively. We use a symbol " $\lceil$ " to refer to a projection of the model on specific entity actions.

In a system with communicating entities like consumers and producers, alongside core exchange actions, we refer to these additional actions as $\mathcal{A}$ such that $P \cup C \subseteq \mathcal{A}$. So, the semantic behavior of the AMQP $\mathscr{R}\mathscr{2}$ protocol is the union of communicating entity actions set $\mathcal{A}$.

Definition 8. (Semantics). $[\![\mathscr{R}\mathscr{2}]\!] = \bigcup_{a \in \mathcal{A}} [\![\mathscr{R}\mathscr{2}_{\lceil a}]\!]$

$$\frac{[\![\mathscr{R}\mathscr{2}_{\lceil p}]\!] = s_1 \xrightarrow{a!\langle k,pld\rangle} s_1' \wedge [\![\mathscr{R}\mathscr{2}_{\lceil Map_k}]\!] = s_2 \xrightarrow{a?\langle k,pld\rangle} s_2'}{\langle s_1, \ldots, s_2, \theta\rangle \xrightarrow{a} \langle s_1', \ldots, s_2', \theta'\rangle} \quad \textit{(Enqueue)}$$

where $\theta' := \theta[Map := enqueue(Map, \langle k, pld\rangle)]$ and $a \in P \cap Map_k$.

$$\frac{[\![\mathscr{R}\mathscr{2}_{\lceil c}]\!] = s \xrightarrow{a?\langle k,pld\rangle} s' \wedge [\![\mathscr{R}\mathscr{2}_{\lceil Map_k}]\!] = s \xrightarrow{a!\langle k,pld\rangle} s'}{\langle s_1, \ldots, s_2, \theta\rangle \xrightarrow{a} \langle s_1', \ldots, s_2', \theta'\rangle} \quad \textit{(Dequeue)}$$

where $\theta' := \theta[Map := dequeue(Map, \langle k, pld\rangle)]$ and $a \in C \cap Map_k$.

Following the communication standard of AMQP, we utilize two distinct operational semantics to accurately represent enqueuing and dequeuing in the AMQP protocol. The first one, labeled as *Enqueue*, is responsible for the process of enqueueing incoming messages. The second one, labeled as *Dequeue*, focuses on dequeuing the message from the queue associated with key k using local channel a . θ is used to evaluate the $\mathscr{R}\mathscr{2}$ queues front values.

4.2. In-Transit payload attack

We utilize the following notations for each message m received at the exchange; the term pld_x represents the erroneous payload of the message transmitted by unauthorized communicating entities denoted as $\mathscr{R}\mathscr{2}_{\lceil \mathscr{A}}$.

Tampering [25] alters messages transmitted by sensors received at the Exchange level. Within the context of the $\mathscr{R}\mathscr{2}$ system, pld is altered during the message transfer. Considering the producer $[\![\mathscr{R}\mathscr{2}_{\lceil p}]\!]$, the exchange $[\![\mathscr{R}\mathscr{2}_{\lceil map_k}]\!]$ and attacker $\mathscr{R}\mathscr{2}_{\lceil \mathscr{A}}$, the tampering is expressed through rules *Success* and *Failure*. Specifically, rule *Success* represents the successful tampering of pld , while rule *Failure* represents the failure to tamper with pld_x. Both operational semantics rules are executed where the message is in *publishing mode*.

For a subscribing entity, the operational semantics rules differ from those of a publisher (as defined in Figure 4). This is because the subscribing entity consumes objects from the queue based on the routing key. In this case, tampering with the payload messages is unnecessary, as the operational semantics rule *Subscribe* in Figure 5 guarantees successful payload consumption.

$$\frac{[\![\mathscr{R}\mathscr{Q}_{\lceil p}]\!] = s_1 \xrightarrow{a!\langle k,pld\rangle}_{1-\lambda} s_1' \wedge [\![\mathscr{R}\mathscr{Q}_{\lceil \mathscr{A}}]\!] = s_2 \xrightarrow{a!\langle k,pld_x\rangle}_{\lambda} s_2' \wedge [\![\mathscr{R}\mathscr{Q}_{\lceil map_k}]\!] = s_3 \xrightarrow{a?\langle k,pld_x\rangle} s_3'}{\langle s_1,\ldots,s_2,\ldots s_3,\theta\rangle \xrightarrow{a}_{\lambda} \langle s_1',\ldots,s_2',\ldots,s_3',\theta'\rangle} \quad \textit{(Success)}$$

where $m = \langle k, pld_x\rangle \ \wedge Map(k) \models \theta$, $\theta' := \theta[map := enqueue(map, m)]$ and $a \in P \cap \mathscr{A} \cap Map_k$.

$$\frac{[\![\mathscr{R}\mathscr{Q}_{\lceil p}]\!] = s_1 \xrightarrow{a!\langle k,pld\rangle}_{1-\lambda} s_1' \wedge [\![\mathscr{R}\mathscr{Q}_{\lceil \mathscr{A}}]\!] = s_2 \xrightarrow{a!\langle k,pld_x\rangle}_{\lambda} s_2' \wedge [\![\mathscr{R}\mathscr{Q}_{\lceil map_k}]\!] = s_3 \xrightarrow{a?\langle k,pld\rangle} s_3'}{\langle s_1,\ldots,s_2,\ldots s_3,\theta\rangle \xrightarrow{a}_{1-\lambda} \langle s_1',\ldots,s_2',\ldots,s_3',\theta'\rangle} \quad \textit{(Failure)}$$

where $m = \langle k, pld\rangle \ \wedge Map(k) \models \theta$, $\theta' := \theta[map := enqueue(map, m)]$, and $a \in P \cap \mathscr{A} \cap Map_k$.

Figure 4: Operational Semantics Rules of the AMQP protocol in Publishing Mode.

$$\frac{[\![\mathscr{R}\mathscr{Q}_{\lceil map_k}]\!] = s_1 \xrightarrow{a!\langle k,pld\rangle} s_1' \wedge [\![\mathscr{R}\mathscr{Q}_{\lceil c}]\!] = s_2 \xrightarrow{a?\langle k,pld\rangle} s_2'}{\langle s_1,\ldots s_2,\theta\rangle \xrightarrow{a} \langle s_1',\ldots,s_2',\theta'\rangle} \quad \textit{(Subscribe)}$$

where $m = \langle k, pld\rangle \quad \theta' := \theta[map := dequeue(map, \langle k, pld\rangle)]$, and $a \in C \cap Map_k$.

Figure 5: Operational Semantics Rules of the AMQP protocol in Subscribe Mode.

5. Transformation-based Reasoning

This section dives into the implementation details of the AMQP protocol in both PRISM and OMNeT++. We then explore how rules are automatically generated based on the execution traces captured by OMNeT++. Finally, we leverage PRISM for formal reasoning about the generated model, which is an abstraction of the initial AMQP implementation.

5.1. The AMQP operational design in PRISM

Within the PRISM model, we differentiate between three types of modules: publisher modules $\mathscr{P}_{\mathscr{M}\lceil p}$ and the exchange module $\mathscr{P}_{\mathscr{M}\lceil map_k}$. Publishers modules provide support for attackers and sensors located at the exchange edge of the AMQP according to the rules *Success* and *Failure*. The $\mathscr{P}_{\mathscr{M}\lceil c}$ is the consumer (i.e., subscriber) module in charge of collecting the data at the edge. In contrast, exchange modules handle enqueueing rule *Enqueue* and dequeuing *Dequeue* operations according to pre-defined operational semantics rules.

The operational semantics rules in Figure 6 governing enqueue and dequeue operations in PRISM are presented in *Enqueue in PRISM* and *Dequeue in PRISM*, respectively. Rule *Enqueue in PRISM* models the act of payloads enqueuing (represented by the command $[\![\mathscr{P}_{\mathscr{M}\lceil p}]\!]$) in the AMQP queue, denoted by $\mathscr{P}_{\mathscr{M}\lceil map_k}$. Upon enqueueing, the update u_2 increments the queue size by one unit. Conversely, rule *Dequeue in PRISM* depicts the dequeue operation, where the update u_2 decrements the value of the variable v_i representing the queue size.

The enqueueing semantic rule is realized through the parallel composition of modules, synchronized by the action a . Formally, this can be expressed as: $\mathscr{P}_{\mathscr{M}} = \mathscr{P}_{\mathscr{M}\lceil p} ||_a \mathscr{P}_{\mathscr{M}\lceil map_k}$. Here, $\mathscr{P}_{\mathscr{M}}$ denotes the overall system, $\mathscr{P}_{\mathscr{M}\lceil p}$ represents the process module, and $\mathscr{P}_{\mathscr{M}\lceil map_k}$ represents the exchange module. The parallel composition is denoted by $||_a$, indicating that the modules synchronize on the action a . Similarly, the dequeuing operation is realized through the parallel composition of modules, synchronized by the action a . This can be formally represented as: $\mathscr{P}_{\mathscr{M}} = \mathscr{P}_{\mathscr{M}\lceil c} ||_a \mathscr{P}_{\mathscr{M}\lceil map_k}$. In this expression, $\mathscr{P}_{\mathscr{M}\lceil c}$ represents the consumer module, which interacts with the exchange module $\mathscr{P}_{\mathscr{M}\lceil map_k}$ through the shared action a .

$$\frac{[\![\mathscr{P}_{\mathscr{M}\lceil p}]\!] = [a]g_1 \to u_1 \wedge [\![\mathscr{P}_{\mathscr{M}\lceil Map_k}]\!] = [a]g_2 \to u_2}{\langle [a]g_1 \to u_1, \ldots, [a]g_2 \to u_2\rangle \xrightarrow{a} \langle u_1, \ldots, u_2\rangle} \quad \textit{(Enqueue in PRISM)}$$

where $\theta' := \theta[u_2[v_i = v_i + 1]]$ and $a \in P \cap Map_k$.

$$\frac{[\![\mathscr{P}_{\mathscr{M}\lceil c}]\!] = [a]g_1 \to u_1 \wedge [\![\mathscr{P}_{\mathscr{M}\lceil Map_k}]\!] = [a]g_2 \to u_2}{\langle [a]g_1 \to u_1, \ldots, [a]g_2 \to u_2\rangle \xrightarrow{a} \langle u_1, \ldots, u_2\rangle} \quad \textit{(Dequeue in PRISM)}$$

where $\theta' := \theta[u_2[v_i = v_i - 1]]$ and $a \in C \cap Map_k$.

Figure 6: Operational Semantics Rules of the AMQP Queuing protocol.

In-transit payload attacks represent the canonical attacks that can occur on payload messages. Rules *Success in PRISM* and *Failure in PRISM* capture the successful and failure tampering of payload message pld , respectively. A probabilistic command implements the attacker's decision-making process based on the attack frequency, denoted by λ. These operational semantics rules formally implement the previously defined attack behavior. However, PRISM commands embody two transitions of the $\mathscr{R}\mathscr{Q}$ that implement the attacker and the producer. In PRISM, both transitions are grouped in one command as portrayed in Figure 7. We use " _ " to model a non selected update on $\mathscr{P}_{\mathscr{M}}$ as u_1 in *Success in PRISM* and u_2 in *Failure in PRISM*. Subscription mode is implemented by the operational semantics rule *Dequeue in PRISM* as equivalent to the operational semantics rule of *Dequeue*

$$\frac{[\![\mathscr{P}_{\mathscr{M}\lceil p}]\!] = [a]g \to \lambda : u_1 + (1-\lambda) : u_2 \wedge [\![\mathscr{P}_{\mathscr{M}\lceil Map_k}]\!] = [a]g_3 \to u_3}{\langle [a]g \to \lambda : u_1 + (1-\lambda) : u_2, \ldots, [a]g_3 \to u_3\rangle \xrightarrow{a}_{1-\lambda} \langle \_, \ldots, u_2, \ldots, u_3\rangle} \quad \textit{(Success in PRISM)}$$

where $\theta' := \theta[u_3[v_i = pld_x]]$, and $a \in P \cap Map_k$.

$$\frac{[\![\mathscr{P}_{\mathscr{M}\lceil p}]\!] = [a]g \to \lambda : u_1 + (1-\lambda) : u_2 \wedge [\![\mathscr{P}_{\mathscr{M}\lceil Map_k}]\!] = [a]g_3 \to u_3}{\langle [a]g \to \lambda : u_1 + (1-\lambda) : u_2; [a]g_3 \to u_3\rangle \xrightarrow{a}_{\lambda} \langle u_1, \ldots, \_, \ldots, u_3\rangle} \quad \textit{(Failure in PRISM)}$$

where $\theta' := \theta[u_3[v_i = pld]]$, and $a \in P \cap Map_k$.

Figure 7: Operational Semantics Rules in PRISM of the AMQP protocol in Publishing Mode

5.2. AMQP in OMNeT++

In this section, our focus will be on the architecture and structure of the Exchange module. In OMNeT++, the keyword "network" is used to describe the architecture, as it comprises instantiated components responsible for implementing the interactions, known as "connections". The code snippet Listing 1 presents a snapshot of a model depicting the architecture of the AMQP. For instance, in lines 4-6, three sensors are instantiated to generate data (a virtual representation of sensors). Line 7 represents the exchange responsible for collecting the sensed data and enqueuing them in their dedicated queue. Line 8 depicts the actuator, which receives data from the Fog defined in line 9. The interaction, as described in section 2.2, is illustrated by a collection of connections in the forms of: $outputgate \to delay \to inputgate$

Considering the architecture depicted in Listing 1, the code snippet Listing 2 provides a snapshot of a C++ implementation of the `Exchange` module. The module is a C++ class that extends the OMNeT class library `cSimpleModule`. Two types of messages, namely `PUSH` and `TRANSMIT`, are triggered by the module. The `PUSH` message triggers the enqueuing operation on the `fifoMap` (line 13), based on the binding keys defined in lines 9-12. The `transmit` message enables the construction of a dedicated structure responsible for transmitting a complex message, as depicted in lines 24-25. The message is then sent through a specific port using the `send()`

Listing 1: An Architecture in OMNeT++

```
network AMQP
{
submodules:
    WL: WaterLevel {}
    RP: RainPrecipitation { }
    WV: WaterVolume { }
    Exchange: Exchange { }
    WF: WaterFlow { }
    Fog: Fog { }
connections:
  RP.data --> {delay = 100ms; } --> Exchange.dataRainPrecipitation;
  WL.data --> {delay = 100ms; } --> Exchange.dataWaterLevel;
  WV.data --> {delay = 100ms; } --> Exchange.dataWaterVolume;

  Exchange.dataToFog --> { delay = 100ms; } --> Fog.dataToFog;
  Fog.fogToData --> {delay = 100ms; } --> Exchange.fogToData;
  Exchange.decisiondataWaterFlow --> {delay = 100ms; } --> WF.decisiondataWaterFlow;
}
```

function in lines 26. The protected function `initialize()` is responsible for initializing the `fifoMap`, while the `handleMessage()` function listens for incoming messages. For example, if a push message is received, the incoming data is enqueued in the `fifoMap`.

Listing 2: A Model of Exchange component

```
class Exchange : public cSimpleModule
{
 public:
        Exchange();
 private:
    cMessage*  PUSH;
    cMessage*  TRANSMIT;
    /* Binding keys*/
    std::string bkWL = "WL";
    std::string bkRP = "RP";
    std::string bkWV = "WV";
    std::string bkWF = "WF";
    std::map<std::string, std::queue<int>> fifoMap;
     ...
    void transmit(int pld);
 protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

void Exchange::transmit(int pld)
{
 std::string result = std::string("WF/") + std::string(std::to_string(pld));
 TRANSMIT = new cMessage(result.c_str());
 send(TRANSMIT, "decisiondataWaterFlow");
}
```

5.3. Traces-driven rules generation for simulation order verification

To enable the construction of an abstract PRISM model from the OMNeT++ execution traces $Seq_{\mathcal{N}\mathcal{D}}$, we have implemented a transformation from a $Seq_{\mathcal{N}\mathcal{D}}$ to $\mathcal{T}$ using PDT and then from $\mathcal{T}$ to $\mathcal{P}_{\widehat{\mathcal{M}}}$ in the PRISM modeling language using a personalized algorithm.

First, building upon the previous definitions in Section 2, the PDT learning $\mathcal{L}: \ Seq_{\mathcal{N}\mathcal{D}} \rightarrow Rule$ is a function taking as input a set of update $[\![Seq_{\mathcal{N}\mathcal{D}}]\!]$ to produce a set of rules as $[\![\mathcal{T}]\!]$:

Definition 9 (Learning)**.** The PDT learning of a set of rules $[\![\mathscr{T}]\!]$ from a sequence of OMNeT++ execution trace $[\![Seq_{\mathscr{N}\mathscr{D}}]\!]$ such that, for an update $t \in [\![Seq_{\mathscr{N}\mathscr{D}}]\!]$ it belongs to a rule $r \in [\![\mathscr{T}]\!]$ such that $r = x_i \overset{d}{\hookrightarrow}_{\lambda} q_i$ where $x_i \models g_i$. g is a produced guard from learning.

> *Note*. We will not delve into the details of the learning function $\mathcal{L}$ in this paper. The learning algorithm is well-established and can be found in the literature ([26, 27]). Readers and practitioners can refer to [28] for a more in-depth explanation.

During the rule generation process using the PDT learning function $\mathcal{L}$, multiple rules (r_i) are generated such that $\forall i \in |\mathscr{T}|,\ r_i \in [\![\mathscr{T}]\!]$ (where $[\![\mathscr{T}]\!]$ denotes the set of rules), so the semantics of the generated $[\![\mathscr{T}]\!]$ is thus the union of the set of rules with regards to collected traces instances $[\![Seq_{\mathscr{N}\mathscr{D}}]\!]$. To express it more clearly, all the generated rules cover the traces with a certain level of accuracy [26, 27] originating from $[\![Seq_{\mathscr{N}\mathscr{D}}]\!]$:

Definition 10. (Semantics). For $t = (s, a, s), [\![\mathscr{T}]\!] = \bigcup_{t \in [\![Seq_{\mathscr{N}\mathscr{D}}]\!]} [\![\mathscr{T}_{\lceil a}]\!]$

In the following, the approach for transforming the generated rules $[\![\mathscr{T}]\!]$ in the modeling language of the PRISM model checker $[\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$ is straightforward. The function $\mathcal{F} : \mathscr{T} \rightarrow \mathscr{P}_{\widehat{\mathscr{M}}}$ is in charge of mapping the generated rules to PRISM commands. When two rules have the same guards leading to different output features, they are merged into single PRISM commands. For instance, assume that $g_1 \rightarrow_{\lambda} q_1$ and $g_1 \rightarrow_{1-\lambda} q_2$, so, the PRISM command takes the form of $[r]g_1 \rightarrow \lambda : q_1 + (1 - \lambda) : q_2$ (detailed in Algorithm 1). In this case, a generated rule r is considered a view of the PRISM model on a specific command $c \in \mathscr{C}$. The semantics of a $[\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$ is thus the union of the behaviors of the views on all generated rules $Rule$.

Definition 11. (Semantics). For $r = (s, d, g, s),\ [\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!] = \bigcup_{r \in [\![\mathscr{T}]\!]} [\![\mathscr{P}_{\widehat{\mathscr{M}}\lceil d}]\!]$

Algorithm 1: PDT rules in $\mathscr{T}$ to PRISM commands in $\mathscr{P}_{\widehat{\mathscr{M}}}$ as $\mathcal{F} : \mathscr{T} \rightarrow \mathscr{P}_{\widehat{\mathscr{M}}}$

Data: A set of rules $[\![\mathscr{T}]\!]$ in a stack $Stack_{\mathscr{T}}$
Result: A set of commands $[\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$ in a stack $Stack_{\mathscr{P}}$

```
1  r ∈ [[T]] ;                                              /* A PDT rule variable */
2  c ∈ [[P_M̂]] ;                                           /* A prism command variable */
3  for Stack_T not empty do
4      r ← Stack_T. pop() ;
5      if Stack_P. find( r. guard()) not empty then
6          c ← Stack_P. find( r . guard()) ;               /* Find a command with a same guard. */
7          Stack_P . update( c, r . prob() , r. output() ) ;   /* Update a command with probability
             and feature output. */
8      else
9          c ← PRISM_Command( r. prob() , r. output()) ;   /* Define a new PRISM command */
10         Stack_P . push( c ) ;                           /* Push the new PRISM command */
11     end
12 end
```

Algorithm 1 details the process of transforming a set of decision tree-generated rules into PRISM commands using the function $\mathcal{F}$. Two variables are defined (lines 1 and 2) to store the current rule and the corresponding PRISM command reference, respectively. The algorithm first pops a rule from the stack containing decision trees ($[\![\mathscr{T}]\!]$) in line 4. It then checks if the rule guard (the condition that activates the rule) already exists in the PRISM command stack $Stack_{\mathscr{P}}$ (line 5). If it does, the existing command is updated to include a reference to the current rule's feature and its occurrence probability (line 7). However, if the guard is not found in the PRISM command stack, a new command is created in line 9 and pushed onto the stack in line 10.

Having established the semantics of generated PDT rules $Rule$ in Definition 10 and the PRISM model in Definition 11 $\mathscr{P}_{\widehat{\mathscr{M}}}$, we advance the following theorem. The generated PRISM model $\mathscr{P}_{\widehat{\mathscr{M}}}$ encapsulates the union of the behaviors of trace instances t within the set $[\![Seq_{\mathcal{ND}}]\!]$.

Lemma 1. *For* $t = (s, a, s), [\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!] = \bigcup_{t \in [\![Seq_{\mathcal{ND}}]\!]} [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil a}]\!]$

Proof. Assume that the command $c \in \mathscr{C}$ covers a finite set of rules $r_0, \ldots, r_n \in [\![\mathscr{T}]\!]$. Following Definition 11 we have $[\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil c}]\!] = [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil d_0}]\!] \cup \cdots \cup [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil d_n}]\!]$. Assume a rule $r_i \in \mathscr{T}$ that covers a sequence of update $t_0, \ldots, t_n \in Seq_{\mathcal{ND}}$. Following Definition 10 we have $[\![\mathscr{T}_{\lceil d_i}]\!] = [\![\mathscr{T}_{\lceil a_0}]\!] \cup \ldots \cup [\![\mathscr{T}_{\lceil a_m}]\!]$. By applying function $\mathcal{F}$ we have $[\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil d_i}]\!] = [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil a_0}]\!] \cup \cdots \cup [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil a_n}]\!] = \bigcup_{t \in [\![Seq_{\mathcal{ND}}]\!]} [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil a}]\!]$ and then for command execution e $[\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil e}]\!] = \bigcup_{t \in [\![Seq_{\mathcal{ND}}]\!]} [\![\mathscr{P}_{\widehat{\mathscr{M}} \lceil a}]\!]$

□

Simulation relations are frequently used to verify that a concrete system faithfully implements an abstract system [29]. In the context of our research paper, we aim to verify not only execution coverage, as proven previously, but also a stronger property: that $\mathcal{ND} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$. First, we portray in Definition 12 the simulation order as a basis of the proof.

Definition 12 (Simulation Order)**.** A simulation for $(\mathcal{ND}, \mathscr{P}_{\widehat{\mathscr{M}}})$ is a binary relation $\mathcal{R} \subseteq \mathcal{ND} \times \mathscr{P}_{\widehat{\mathscr{M}}}$ such that:

1. $\forall s_1 \in \overline{X}_{\mathcal{ND}} \bullet \exists s_2 \in \overline{X}_{\mathscr{P}_{\widehat{\mathscr{M}}}}$
2. for all $(s_1, s_2) \in \mathcal{R}$) it holds that:
   (a) L(s_1)= L(s_2)
   (b) if there is a transition $s_1 \to s_1'$ then there is a transition $s_2 \to s_2'$ with $(s_1, s_2) \in \mathcal{R}$.

We can establish the following theorem based on Definition 12.

Lemma 2. *$\mathcal{ND}$ is simulated by $\mathscr{P}_{\widehat{\mathscr{M}}}$ denoted by $\mathcal{ND} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$, if there exists a simulation $\mathcal{R}$ for $(\mathcal{ND}, \mathscr{P}_{\widehat{\mathscr{M}}})$.*

Proof. The condition (1) in Definition 12 is verified since all the initial value in $s_1 = \langle x_0, \ldots, x_n \rangle$ are satisfied by the initial value of $s_2 = \langle x_0, \ldots, x_n \rangle$. The condition (2) in Definition 12 is also verified as all the transitions $t \in [\![Seq_{\mathcal{ND}}]\!]$ are covered by the commands execution in $[\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$ according to Lemma 1.

□

5.4. Abstraction soundness and property preservation

Simulation relations serve as a foundation for abstraction techniques. In these techniques, the model to be verified is replaced by a smaller, abstract model. The verification effort then focuses on the abstract model, rather than the original one.

We have built the OMNeT++ model $\mathcal{ND}$ that captures the same structure and commands of the operational semantics rules *Dequeue* and *Enqueue* in Section 4.1 as well as the PRISM model $\mathscr{P}_{\mathscr{M}}$. According to the Definition 13 based on [30] we say that both model $\mathscr{P}_{\mathscr{M}}$ and $\mathcal{ND}$ are comparable.

Definition 13 (Comparability)**.** Two models $\mathscr{P}_{\mathscr{M}}$ and $\mathcal{ND}$ are comparable if they have identical sets of variables and actions.

The models $\mathscr{P}_{\mathscr{M}}$ and $\mathcal{ND}$ exhibit strong comparability. This is because they share the same set of variables. Additionally, the actions in $\mathscr{P}_{\mathscr{M}}$ correspond directly to the gates in $\mathcal{ND}$. So, for a pair of strongly comparable $\mathscr{P}_{\mathscr{M}}$ and $\mathcal{ND}$, we say that $\mathcal{ND} \preceq \mathscr{P}_{\mathscr{M}}$ and $\mathscr{P}_{\mathscr{M}} \preceq \mathcal{ND}$. In this case, we consider that both models satisfy the simulation equivalence (i.e., The simulation equivalence is not addressed in this paper since it is irrelevant to the research purposes). Considering the relation between different model $\mathscr{P}_{\mathscr{M}}$, $\mathscr{P}_{\mathcal{ND}}$, and $\mathscr{P}_{\widehat{\mathscr{M}}}$, Based on Lemma 3, we can establish reasoning for the abstraction relation between models.

Lemma 3 (Abstraction). *For* $\mathscr{P}_{\mathscr{M}}$ *as the concrete PRISM model and* $\mathscr{P}_{\widehat{\mathscr{M}}}$ *as an abstract model then if there is a simulation relation* $\mathcal{R}_S$ *from* $\mathscr{P}_{\mathscr{M}}$ *to* $\mathscr{P}_{\widehat{\mathscr{M}}}$.

Proof. From Definition 13, we have already proved that $\mathscr{P}_{\mathscr{M}} \preceq \mathscr{ND}$ and from Lemma 2, we have proved that $\mathscr{ND} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$, so, by induction $\mathscr{P}_{\mathscr{M}} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$ and $\mathscr{P}_{\widehat{\mathscr{M}}}$ is an abstraction of $\mathscr{P}_{\mathscr{M}}$. □

In Lemma 4, we present the soundness of $\mathcal{R}_S$ abstraction relation.

Lemma 4 (Abstraction Soundness). $\mathscr{P}_{\mathscr{M}} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$ *implies that* $[\![\mathscr{P}_{\mathscr{M}}]\!] \subseteq [\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$.

Proof. Assume that $\mathscr{P}_{\mathscr{M}} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$. Let $\pi_1 \in [\![\mathscr{P}_{\mathscr{M}}]\!]$ where $s_1 = \langle x_0, \ldots, x_n \rangle$ and $s_1 \in \overline{X}_{\mathscr{P}_{\mathscr{M}}}$. Since $\mathscr{P}_{\mathscr{M}} \preceq \mathscr{P}_{\widehat{\mathscr{M}}}$, there exists $s_2 \in \overline{X}_{\mathscr{P}_{\widehat{\mathscr{M}}}}$ such that $s_1 \preceq s_2$. While considering the simulation relation $\mathcal{R}$ as in [29], and π_1 as execution path from s_1 then there is an execution path π_2 such that π_1 and π_2 are statewize $\mathcal{R}$ -related. This yields that $[\![\mathscr{P}_{\mathscr{M}_{\lceil \pi_1}}]\!] = [\![\mathscr{P}_{\widehat{\mathscr{M}}_{\lceil \pi_2}}]\!]$. □

Proposition 5 (Property Preservation). *For two PRISM models* $\mathscr{P}_{\mathscr{M}}$ *and* $\mathscr{P}_{\widehat{\mathscr{M}}}$ *if* $\mathscr{P}_{\widehat{\mathscr{M}}} \models \phi$ *implies* $\mathscr{P}_{\mathscr{M}} \models \phi$.

To prove Proposition 5, we have to provide a definition of property satisfaction in [29] as follows:

Definition 14 (Satisfaction relation for ϕ). For a model $\mathscr{P}_{\widehat{\mathscr{M}}}$, $\mathscr{P}_{\widehat{\mathscr{M}}}$ $satisfies$ ϕ , denoted $\mathscr{P}_{\widehat{\mathscr{M}}} \models \phi$ iff $[\![\mathscr{P}_{\widehat{\mathscr{M}}_{\lceil \pi_2}}]\!] \subseteq \phi$.

Proof. Assume $[\![\mathscr{P}_{\mathscr{M}}]\!] \subseteq [\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$, and let ϕ be a PCTL property such that $\mathscr{P}_{\widehat{\mathscr{M}}} \models \phi$. From Definition 14 it follows that $[\![\mathscr{P}_{\widehat{\mathscr{M}}_{\lceil \pi_2}}]\!] \subseteq \phi$. Given $[\![\mathscr{P}_{\mathscr{M}}]\!] \subseteq [\![\mathscr{P}_{\widehat{\mathscr{M}}}]\!]$, it now follows that $\mathscr{P}_{\mathscr{M}} \models \phi$. By Definition 14, it follows that $\mathscr{P}_{\mathscr{M}} \models \phi$. □

6. Evaluation

This research introduces a novel approach that leverages a combination of simulation and machine learning to address the complexity challenges of formal modeling, particularly the state explosion problem. We present an experimental evaluation applied to a concrete use case related to a water dam infrastructure and compare the two approaches' capabilities in incorporating functional and security properties for analysis. We consider three research questions in this analysis, where RQ1 and RQ2 focus on a modifying message exchange between communicating entities on the AMQP and RQ3 addresses the feasibility of modeling the system in PRISM $\mathscr{P}_{\mathscr{M}}$ and the learned model $\mathscr{P}_{\widehat{\mathscr{M}}}$.

RQ1: Given a message received at the exchange level, what is the probability that the message eventually will not be tampered with regarding the queue size?

RQ2: What is the cumulative number of tampered messages encountered at the exchange level regarding the queue size?

RQ3: What is the relative efficiency of model checking properties related to RQ1 and RQ2 on the initial PRISM model $\mathscr{P}_{\mathscr{M}}$ compared to the learned model $\mathscr{P}_{\widehat{\mathscr{M}}}$ (in the time of constructing and model checking)?

Experimental setup.. We leverage the PRISM model checker (v4.7) for formal verification. Experiments are executed on an Ubuntu system with an I7 processor and 32GB of RAM. The system model, adhering to PRISM's probabilistic automata formalism, is accessible through the reference provided in [31]. While PRISM supports various engines (refer to documentation [32]) offering performance benefits for specific model structures, the "MTBDD" demonstrated the most effectiveness in providing results during our experiments. We extract parameters associated with data manipulation attacks from a dataset and utilize a custom Python script to calculate attack rates. Properties of interest are expressed in PCTL for model checking.

Artifacts.. The experiments detailed in this section are publicly available and fully reproducible. The PRISM and OMNeT++ source codes can be accessed from the public GitHub repository [31].

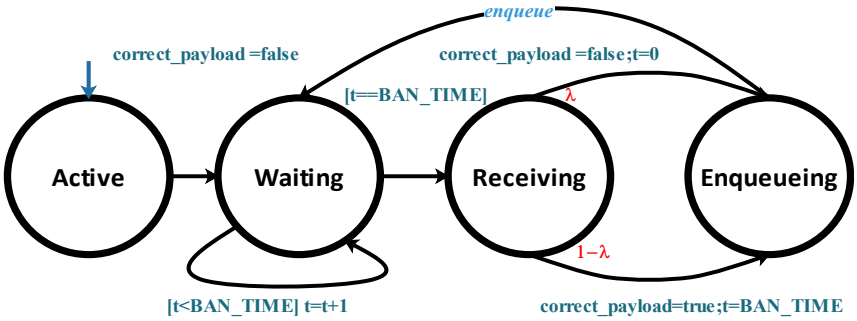

Figure 8: Sensor-side Module.

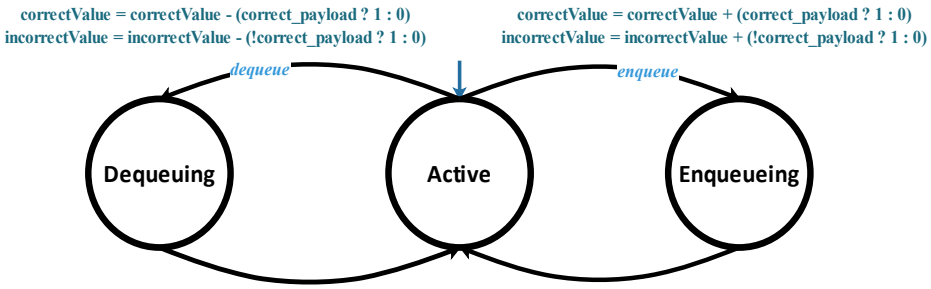

Figure 9: Exchange Module.

6.1. Analyzed operational design

The model, aligning with the use case in the section 3 and the reference standard [16], provides context for the experiments. It represents a system with three sensors and one actuator connected through a gateway utilizing a broker (AMQP). The specific functionalities of the sensors and actuators are intentionally abstracted away, focusing solely on the core-side implementation at a high level. For example, the sensor module (see Figure 8) is modeled with four states: Active, Waiting, Receiving, and Enqueuing. The Active state initiates model execution. In the Waiting state, the system remains idle, awaiting either sensor data or the expiration of the banning timer. Upon receiving data or the timer expiring, the system moves to the Receiving state. Here, it determines the data's quality (correct or incorrect) based on the attack rate. If valid, the banning timer is reset to zero. Finally, the validated data is pushed to the queue in the Enqueuing state.

To represent each sensor, we replicate the sensor module (i.e., publishers). PRISM's renaming capabilities allow this duplication while maintaining distinct functionalities. However, for clarity and to avoid potential experimental errors, we opted to write separate modules for each sensor. This enhances model understanding by readers. Next, we model the exchange module responsible for queuing and dequeuing payload messages. Synchronization mechanisms ensure adherence to the previously defined read/write operational semantics (see section 5.1). The enqueuing and dequeuing behavior reflects the mapping of operational semantics rules to PRISM constructs, as detailed in the transformation section. We forgo modeling the binding key that associates queues with specific payloads for simplicity. Instead, we directly link each sensor's module with the exchange module handling its respective queue. This simplified approach can be replicated for each additional sensor.

Modeling the Exchange module (see Figure 9) involves three states: Active, Enqueueing, and Dequeuing. In the Active state, the exchange can perform two operations: enqueue a new value into the queue and dequeue a value based on the availability of consumers and producers. Synchronization is performed in case of these availabilities.

6.2. Traces collection procedure

During the rule generation process with PDT, four rules, r_1 , r_2 , r_3 and r_4 , were generated. The first two rules, r_1 , $r_2 \in [\![Rule]\!]$.

$$(v_{wl} = 1) \wedge (v_{wv} = 1) \wedge (v_{wh} = 1) \wedge (v_{at} = 0) \xhookrightarrow{r_1}_{0.17} (v_{wf} = 0)$$
$$(v_{wl} = 1) \wedge (v_{wv} = 1) \wedge (v_{wh} = 1) \wedge (v_{at} = 0) \xhookrightarrow{r_2}_{0.17} (v_{wf} = 1)$$

These rules r_1 , r_2 indicate that when all sensor values $v_{wl}, v_{wv} v_{wh}$ are available and no attacks are detected ($v_{at} = 0$), there is a 0.17 probability of identifying an incorrect water flow ($v_{wf} = 0$) and a 0.83 probability of identifying a correct water flow ($v_{wf} = 1$). In contrast, when attacks are detected ($v_{at} = 1$) , the following two rules are identified:

$$(v_{wl} = 1) \wedge (v_{wv} = 1) \wedge (v_{wh} = 1) \wedge (v_{at} = 1) \xhookrightarrow{r_3}_{0.85} (v_{wf} = 0)$$
$$(v_{wl} = 1) \wedge (v_{wv} = 1) \wedge (v_{wh} = 1) \wedge (v_{at} = 1) \xhookrightarrow{r_4}_{0.15} (v_{wf} = 1).$$

These rules r_3 , r_4 signify that the probability of identifying an incorrect water flow increases to 0.85, while the probability of identifying a correct water flow decreases to 0.15 when attacks are present.

Listing 3: A PRISM Model of the Generated Rules

```
1  const int AT;
2  module GeneratedDecisiontree
3      RP : [0..1] init 1 ;
4      WV : [0..1] init 1 ;
5      WL : [0..1] init 1 ;
6      WF : [-1..1] init -1 ;
7    [rule12] RP=1 & WV=1 & WL=1 & AT=0 -> 0.16706586826347307:(WF'=0)+ 0.8329341317365269:(WF'=1);
8    [rule34] RP=1 & WV=1 & WL=1 & AT=1 -> 0.8450901803607214:(WF'=0)+ 0.15490981963927857:(WF'=1);
9  endmodule
```

For that, we consider again the use case in Figure 3. Its transformation is shown in Listing 3. A block of the generated dataset corresponds to a PRISM `module`. A bounded integer variables are created in lines 3-6. Finally, each generated rule is transformed into a guarded command (refer to Section 2.1 for details). In this configuration, the guard condition in the command represents the source state. The update part assigns the successor state to a local variable. However, the generated rules may contain the same source state but have different successor states with assigned probabilities. In these cases, we merge the rules into a single probabilistic command. For example, the rules r_1 and r_2 are merged to the probabilistic command in line 7 whereas the rules r_3 and r_4 are merged to the probabilistic command in line 8.

6.3. Results

We use the PRISM model checker to verify the functional properties expressed as RQ1 and RQ2. and then evaluate the experiments concerning RQ3.

6.3.1. Model checking.

Before using the PRISM model checker, we need to translate the properties we want to verify, derived from the research questions, into the PCTL language. For example, Research Question 1 (RQ1) asks whether eventual tampering with a message in the queue can occur. To express this in PCTL, we would write a formula that specifies the system should eventually reach a state where tampering has happened. It is expressed as PRO1.

PRO1

$$\texttt{P} =?[\texttt{G}(!(\text{“}\texttt{tampered\_payload}\text{”}) \Rightarrow \texttt{F}(\text{“}\texttt{tampered\_payload}\text{”}))]$$

This property essentially expresses that if the message payload starts untampered (! “tampered_payload”), then eventually it will be tampered with (“tampered_payload”) at some point in the system's execution. We use, Globally (G) which indicates that the property needs to hold at all points throughout the system's execution. The label “tampered_payload” refers to a PRISM label that captures the tampering of a publisher's device's payload. It is expressed as follows:

$$label\ \text{”}tampered\_payload\text{”} = wv\_tampered\_payload| \\ wl\_tampered\_payload|rp\_tampered\_payload;$$

$wv\_tampered\_payload$, $wl\_tampered\_payload$, and $rp\_tampered\_payload$ refers to a variable that captures tampering specifically at the exchange module.

Listing 4: A Reward Structure

```
1  rewards "tamperred_consumed"
2      [wf_enqueue] wv_correct_payload: 1;
3      [rp_enqueue] rp_correct_payload: 1;
4      [wl_enqueue] wl_correct_payload: 1;
5  endrewards
```

Similar to RQ1, RQ2 can be translated into a PCTL property that specifies the cumulative cost associated with the buffer size. To achieve this, we need to augment the model with a reward structure. PRISM rewards are real values assigned to specific states or transitions in the model. In our case, these rewards are synchronized with the “enqueue” action on the exchange module transition as in Listing 4. Lines 2-4 define action labels that correspond to a reward structure. These rewards are only synchronized with exchange commands if an incorrect payload has been collected.

Reachability reward properties are used to analyze systems based on the accumulated reward along paths in a model. These properties associate a reward with each path and evaluate whether the total reward reaches a specific value. In contrast, cumulative reward properties also associate a reward with each path in a model, but they only consider the accumulated reward up to a specific time-bound. This time-bound, denoted by T, must be a non-negative integer. Our research question RQ2 is expressed as a cumulative reward property, as shown in PRO2.

PRO2

$$\texttt{R}\{\text{“}\texttt{tampered\_consumed}\text{”}\} =?[\texttt{C}^{\leq \texttt{T}}]$$

The verification of property PRO1 indicates a high (99.99%) probability of message tampering during the exchange between sensors and the queues. Additionally, Figure 10 (green line) visually confirms a significant increase in the expected number of message modifications within a 100-unit timeframe. Although the rate remains stable with a buffer size exceeding 4, this increase likely results from the high attack frequency observed in the dataset. This suggests the attack overwhelms even the message expiry functionality of the AMQP [16], leading to a higher proportion of corrupted messages.

6.3.2. Scalability.

We evaluate our abstraction's efficiency and effectiveness by verifying PCTL properties on both the original and the abstract models. To this end, we compare the results of the verification cost (β) and the abstraction

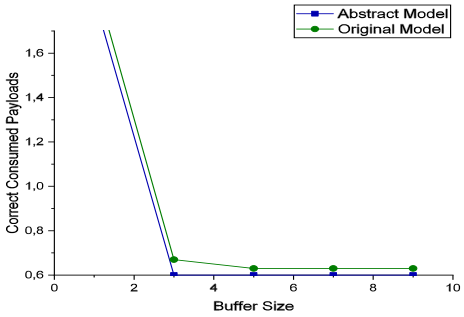

Figure 10: Expected of Consumed Correct Payloads.

efficiency (η) [33, 34]. To evaluate the verification cost, we measure the time required for verifying a given property, denoted by T_v . And, to evaluate the abstraction efficiency, we measure the time required to construct the model T_c such that $\beta = 1 - \frac{T_v(\mathscr{P}_{\widehat{\mathscr{M}}})}{T_v(\mathscr{P}_{\mathscr{M}})}$ and $\eta = 1 - \frac{T_c(\mathscr{P}_{\widehat{\mathscr{M}}})}{T_c(\mathscr{P}_{\mathscr{M}})}$.

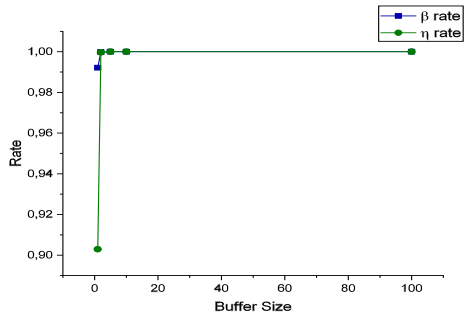

Figure 11: Performances for Verification of PRO1.

We present experiments on model construction and checking for properties PRO1 and PRO2, computing construction and verification rate β and η , respectively. The results are shown in Figure 11 and Figure 12. These figures demonstrate the high performance of the abstraction technique in terms of both construction time and verification time. However, verification for properties PRO1 and PRO2 did not yield results for buffer sizes of 3 elements or more due to the well-known state-space explosion problem. Limits become particularly useful when we want to analyze the function's behavior at extreme values of $T_c(\mathscr{P}_{\mathscr{M}})$ and $T_v(\mathscr{P}_{\mathscr{M}})$, especially when they approaches positive infinity. In this case, the limit would approach 1 because the constant terms ($T_c(\widehat{\mathscr{M}})$ and $T_v(\widehat{\mathscr{M}})$) become insignificant compared to the infinitely large value of $T_c(\mathscr{P}_{\mathscr{M}})$ and $T_v(\mathscr{P}_{\mathscr{M}})$ in the denominator. *These results address research question RQ3, demonstrating that the abstract model outperforms the original model.*

6.4. Threats to validity

The current formalism employed to capture the AMQP protocol might be susceptible to limitations when attempting to encompass the broader range of communication schemas supported by the AMQP: Fanout, Topic, and Headers. This potential inconsistency necessitates an investigation into two primary courses of action: either enhancing the existing model with additional constructs or exploring the adoption of a more suitable formalism from the established literature.

The modeled system currently simulates a single exchange node. However, the AMQP allows for the use of multiple exchanges, which can improve load balancing and potentially reduce susceptibility to errors and attack frequencies.

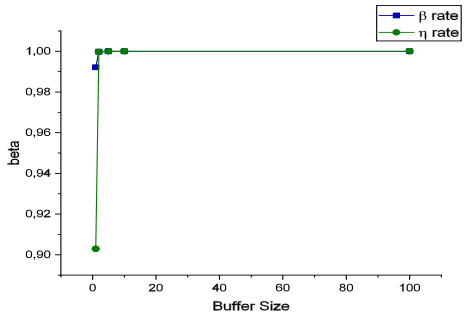

Figure 12: Performances for Verification of PRO2.

OMNeT++ is selected for our experimentation due to its widespread adoption in studies related to wireless networks [35], IoT [36], vehicular communication [37], and many other networking domains. Moreover, finding support and collaborating with experts in the field is more accessible. However, we had to include C/C++ functions to collect the component's execution traces.

The execution traces are generated by a simulator rather than a real infrastructure, which means that the collection process may not accurately capture the relevant parameters to the actual environment. Consequently, it becomes necessary to handle outliers and eliminate meaningless data.

6.5. Discussion

We have showcased the practicality of formal methods, particularly model checking of PRISM models, in the modeling and verifying of the AMQP communication protocol on direct message exchange. We encoded the relative functional and non-functional requirements using PCTL and formally verified them. In Section 4, we formalize rules on tampering during the transmission of payload messages. However, as evidenced by the experiment, it should be noted that the verification process becomes hard when models incorporate a significant number of variables with a large buffer size. This is attributed to algorithms explicitly designed for verifying liveness properties [24]. In scenarios like this, abstraction can prove advantageous in reducing computational overhead, yet it may result in a loss of model meaning such as losing the buffer structure. One of the solutions we propose is to employ OMNet++ to model the system and subsequently learn PRISM models. These models incorporate statistical observations from the streaming simulation dataset, resembling the initial PRISM model regarding verification results. However, verifying the abstract model yields results closely matching the original model (see Figure 10, blue line).

From an engineering perspective, the model transformation has a high degree of automation. The generation artifact consumes the dataset and subsequently produces decision trees that are explored to generate the PRISM commands. This automated process facilitates replication across various projects aiming to formally verify and validate communication protocols.

7. Related Work

The AMQP protocol has been implemented by RabbitMQ and ActiveMQ brokers. It has been validated and verified in numerous studies [38, 39, 40, 41, 42]. The paper [38] presents a formal verification of the RabbitMQ protocol. By modeling RabbitMQ with timed automata and utilizing the UPPAAL model checker [43], essential properties such as Reachability of Data and Message Acknowledgement are verified, demonstrating that RabbitMQ satisfactorily meets these properties. The paper [39] compares the performance of RabbitMQ and ActiveMQ brokers, explicitly examining the sending and receiving of messages. The results reveal differences between the two brokers: the performance analysis demonstrates that ActiveMQ excels in message reception speed. At the same time, RabbitMQ outperforms delivering messages from the broker to the client, thanks to its implemented security protocol at the reception stage.

The research in [41] compares the performance of two popular message brokers, RabbitMQ and Apache Kafka, in the context of fog computing. It highlights the limitations of using REST API for communication in IoT due to synchronous communication and proposes using a message broker to improve reliability. The test results demonstrate that Apache Kafka has higher throughput for calculable message sizes, while RabbitMQ performs better for numerous message sizes. The research in [42] explores the design considerations for scalability and high availability, discusses the usage of clustered RabbitMQ nodes and mirrored queues, and presents the results of performance by simulation tests.

While examining the related literature, we found that the work conducted by [38] focuses on functional properties using UPPAAL [43]. Studies such as [39, 40, 41, 42] primarily employ simulation techniques. However, these studies do not consider the modeling of attacks within the system, which is a crucial aspect given the complexity of the architecture. Our paper, in contrast, offers a comprehensive analysis of attacks across two distinct paths, investigating how to bridge the gap between the formal verification techniques employed by researchers and the practical expertise of simulation practitioners.

8. Conclusion

This study introduces a novel approach that leverages the combined strengths of probabilistic automata (PA) and OMNeT++ to model the behavior of the AMQP protocol implemented by more sophisticated and recent communication and architecture middleware such as RabbitMQ. This approach specifically targets potential data corruption attacks. By employing both PRISM for formal analysis and OMNeT++ for simulation, the study offers a comprehensive evaluation of security vulnerabilities in modern distributed system architectures. Validation is achieved through collected OMNeT++ simulation traces. These traces are used to learn the PA model and investigate potential threats to the payload data within a water dam infrastructure scenario. The study highlights the potential reduction of formal modeling complexity through abstraction and the scalability benefits of using OMNeT++ for modeling and PDT for learning PA models. Future work will explore the integration of additional exchange modules to model composability between generated PDTs in PRISM.

References

[1] S. Chaudhary, P. K. Mishra, Ddos attacks in industrial iot: A survey, Computer Networks 236 (2023) 110015. URL: https://www.sciencedirect.com/science/article/pii/S1389128623004607. doi:https://doi.org/10.1016/j.comnet.2023.110015.

[2] Q. Rouland, B. Hamid, J. Jaskolka, Reusable formal models for threat specification, detection, and treatment, in: Reuse in Emerging Software Engineering Practices, Springer International Publishing, Cham, 2020, pp. 52–68.

[3] R. K. Shrivastava, S. P. Singh, M. K. Hasan, Gagandeep, S. Islam, S. Abdullah, A. H. M. Aman, Securing internet of things devices against code tampering attacks using return oriented programming, Computer Communications 193 (2022) 38–46.

[4] O. M. T. Committee, Mqtt: A lightweight messaging protocol for small sensors and mobile devices, OASIS Standard (2014).

[5] Z. Shelby, K. Hartke, Coap: An application protocol for resource-constrained iot devices, in: Proceedings of the 8th International Conference on Embedded Networked Sensor Systems, ACM, 2010.

[6] A. Baouya, O. Ait Mohamed, D. Bennouar, S. Ouchani, Safety analysis of train control system based on model-driven design methodology, Computers in Industry 105 (2019) 1–16. URL: https://www.sciencedirect.com/science/article/pii/S0166361518302276. doi:https://doi.org/10.1016/j.compind.2018.10.007.

[7] A. Fortas, E. Kerkouche, A. Chaoui, Formal verification of iot applications using rewriting logic: An mde-based approach, Science of Computer Programming 222 (2022) 102859. URL: https://www.sciencedirect.com/science/article/pii/S0167642322000922. doi:https://doi.org/10.1016/j.scico.2022.102859.

[8] E. Dubois, C. Bortolaso, D. Appert, G. Gauffre, An mde-based framework to support the development of mixed interactive systems, Science of Computer Programming 89 (2014) 199–221. URL: https://www.sciencedirect.com/science/article/pii/S0167642313000671. doi:https://doi.org/10.1016/j.scico.2013.03.007, special issue on Success Stories in Model Driven Engineering.

[9] Önder Babur, A. Suresh, W. Alberts, L. Cleophas, R. Schiffelers, M. van den Brand, Chapter 11 - model analytics for industrial mde ecosystems, in: B. Tekinerdogan, Önder Babur, L. Cleophas, M. van den Brand, M. Akşit (Eds.), Model Management and Analytics for Large Scale Systems, Academic Press, 2020, pp. 273–316. URL: https://www.sciencedirect.com/science/article/pii/B9780128166499000211. doi:https://doi.org/10.1016/B978-0-12-816649-9.00021-1.

[10] M. Brambilla, J. Cabot, M. Wimmer, Model-driven Software Engineering in Practice, Synthesis digital library of engineering and computer science, Morgan & Claypool, 2012. URL: https://books.google.fr/books?id=2tVu-wC4XkAC.

[11] A. Baouya, S. Chehida, S. Bensalem, L. Gurgen, R. Nicholson, M. Cantero, M. Diaznava, E. Ferrera, Deploying warehouse robots with confidence: the BRAIN-IoT framework's functional assurance, Journal of Supercomputing 80 (2024) 1206–1237. URL: https://link.springer.com/10.1007/s11227-023-05483-x. doi:10.1007/s11227-023-05483-x.

[12] B. L. Mediouni, I. Dragomir, A. Nouri, S. Bensalem, Model-based design of resilient systems using quantitative risk assessment, Innovations in Systems and Software Engineering 20 (2024) 3–16. URL: https://doi.org/10.1007/s11334-023-00527-0. doi:10.1007/s11334-023-00527-0.
[13] A. Baouya, O. A. Mohamed, S. Ouchani, D. Bennouar, Reliability-driven automotive software deployment based on a parametrizable probabilistic model checking, Expert Systems with Applications 174 (2021) 114572. URL: https://www.sciencedirect.com/science/article/pii/S0957417421000130. doi:https://doi.org/10.1016/j.eswa.2021.114572.
[14] A. Baouya, S. Ouchani, S. Bensalem, Formal modelling and security analysis of inter-operable systems, volume 13343, Springer International Publishing, ????, pp. 555–567.
[15] A. Baouya, B. Hamid, L. Gürgen, S. Bensalem, Rigorous security analysis of rabbitmq broker with concurrent stochastic games, Internet of Things 26 (2024) 101161. URL: https://www.sciencedirect.com/science/article/pii/S2542660524001021. doi:https://doi.org/10.1016/j.iot.2024.101161.
[16] S. Aiyagari, A. Richardson, M. Arrott, M. Ritchie, M. Atwell, Advanced message queuing protocol specification, ???? Retrieved from RabbitMQ https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf.
[17] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of *LNCS*, Springer, 2011, pp. 585–591.
[18] OMNeT++, Discrete Event Simulator, https://omnetpp.org/, Since 2008. [Accessed: January-2023].
[19] M. Kwiatkowska, G. Norman, D. Parker, Quantitative analysis with the probabilistic model checker prism, Electronic Notes in Theoretical Computer Science 153 (2006). Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).
[20] M. L. Wei Zhou, Yi Lu, K. Zhang, Machine learning methods based on probabilistic decision tree under the multi-valued preference environment, Economic Research-Ekonomska Istraživanja 35 (2022) 38–59. doi:10.1080/1331677X.2021.1875866.
[21] T. C. R. Community, CAPEC - Attack Pattern ID: 384, https://capec.mitre.org/data/definitions/384.html, 2018. [Accessed: January-2023].
[22] Brain-iot, https://www.brain-iot.eu/, 2020.
[23] Kentyou, Sensinact delivrables. Brain-IoT D4.5: Final Deployment and Operation Enablers, ????
[24] R. Alur, Principles of Cyber-Physical Systems, The MIT Press, 2015.
[25] A. Shostack, Threat Modeling: Designing for Security, Wiley, 2014.
[26] P.-N. Tan, M. Steinbach, A. Karpatne, V. Kumar, Introduction to Data Mining (2nd Edition), 2nd ed., Pearson, 2018.
[27] T. Hastie, R. Tibshirani, J. Friedman, Additive Models, Trees, and Related Methods, Springer New York, New York, NY, 2009, pp. 295–336. URL: https://doi.org/10.1007/978-0-387-84858-7_9. doi:10.1007/978-0-387-84858-7_9.
[28] L. Rokach, O. Maimon, Data Mining with Decision Trees, 2nd ed., WORLD SCIENTIFIC, 2014. URL: https://www.worldscientific.com/doi/abs/10.1142/9097. doi:10.1142/9097. arXiv:https://www.worldscientific.com/doi/pdf/10.1142/9097.
[29] C. Baier, J.-P. Katoen, Principles of model checking, The MIT Press, ????
[30] S. Mitra, Verifying Cyber-Physical Systems: A Path to Safe Autonomy, CRC Press, 2021.
[31] **, Paper Artefacts Sources, https://blindreviewforwcj.github.io/models2024.html, ????
[32] P. D. T. (eds.), PRISM Manual, ???? URL: https://www.prismmodelchecker.org/manual/ConfiguringPRISM/ComputationEngines, accessed: March 27, 2024.
[33] S. Ouchani, O. Aït Mohamed, M. Debbabi, A property-based abstraction framework for sysml activity diagrams, Knowledge-Based Systems 56 (2014) 328–343.
[34] C. Wang, G. D. Hachtel, F. Somenzi, Abstraction Refinement for Large Scale Model Checking, Integrated Circuits and Systems, Springer, 2006.
[35] AVENS, Aerial VEhicle Network Simulator , https://www.lsec.icmc.usp.br/en/avens, Since 2013. [Accessed: 2023].
[36] COSSIM, Cyber-Physical Systems (CPS) Simulator Framework, https://cossim.org/, Since 2017. [Accessed: 2023].
[37] Artery, Artery V2X Simulation Framework, http://artery.v2x-research.eu/, Since 2014. [Accessed: 2023].
[38] R. Li, J. Yin, H. Zhu, Modeling and analysis of rabbitmq using uppaal, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 79–86. doi:10.1109/TrustCom50675.2020.00024.
[39] V. M. Ionescu, The analysis of the performance of rabbitmq and activemq, in: 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER), 2015, pp. 132–137. doi:10.1109/RoEduNet.2015.7311982.
[40] X. J. Hong, H. Sik Yang, Y. H. Kim, Performance analysis of restful api and rabbitmq for microservice web application, in: 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018, pp. 257–259. doi:10.1109/ICTC.2018.8539409.
[41] A. E. Bagaskara, S. Setyorini, A. A. Wardana, Performance analysis of message broker for communication in fog computing, in: 2020 12th International Conference on Information Technology and Electrical Engineering (ICITEE), 2020.
[42] M. Rostanski, K. Grochla, A. Seman, Evaluation of highly available and fault-tolerant middleware clustered architectures using rabbitmq, in: 2014 Federated Conference on Computer Science and Information Systems, 2014, pp. 879–884. doi:10.15439/2014F48.
[43] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, M. Hendriks, Uppaal 4.0 (2006).