Modeling Energy-Performance-Aware IoT Networks Using Hierarchical Colored Petri Nets

Diogo Lages* ‡, Marcos Falcão*, Andson Balieiro*, Jamilson Dantas*, Dalton Valadares†
*Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife, Brazil.
‡ Universidade Federal do Agreste de Pernambuco (UFAPE), Garanhuns, Brazil.
† Colégio Agrícola Vidal de Negreiros (CAVN), Universidade Federal da Paraíba (UFPB), Bananeiras, Brazil.
{dll, mrmf, amb4, jrd}@cin.ufpe.br, dalton.valadares@embedded.ufcg.edu.br

***Abstract*—Modern Internet of Things (IoT) networks comprise a vast number of interconnected devices capable of seamless communication and cooperation, thereby enhancing the intelligence and efficiency of different environments. They are growing in scale and complexity, making it difficult to assess energy-performance trade-offs before deployment. Yet robust pre-deployment evaluation is critical to ensure energy efficiency and compliance with performance requirements. To address this challenge, we introduce a hierarchical Colored Petri Net (CPN) formal model that jointly captures both device-level energy usage and network performance metrics. The proposed model integrates CPU power states, transceiver operations, communication delays, and unreliable channel conditions into a unified framework. Key innovations include support for heterogeneous CPU architectures and multiple transceivers per node, bidirectional communication, and fine-grained interactions between CPU and transceiver components. This fine-grained formal approach enables combined evaluation of energy consumption and throughput under realistic conditions before physical implementation. We validate the model against real hardware, comprising an ARM7TDMI CPU and an SX1278 LoRa transceiver, demonstrating high accuracy with relative errors below 0.4% for CPU energy, under 2% for transceiver energy, and near-zero for throughput. These results highlight the model's accuracy and practical relevance. Overall, the work contributes a unified, modular, and formally verifiable modeling framework to enhance energy-performance analysis for complex, realistic IoT systems.**

***Index Terms*—IoT Networks, Hierarchical Coloured Petri Nets, Energy-Performance Evaluation.**

I. Introduction

The Internet of Things (IoT) paradigm integrates a vast number of interconnected devices capable of seamless communication and cooperation, thereby enhancing the intelligence and efficiency of different environments [1]. IoT systems comprise heterogeneous, battery-powered devices that perform sensing, control, and communication tasks with minimal human intervention [2]. The adoption of IoT devices has increased sharply in recent years, reaching approximately 18.24 billion active devices in 2024, with projections indicating more than 55 billion connected devices by 2035, spanning applications such as smart homes, smart cities, intelligent agriculture, traffic monitoring, and supply chain management [3]. As IoT networks continue to expand in scale and complexity and are increasingly deployed in remote or hostile environments [4], the need for robust pre-deployment evaluation techniques becomes critical to ensure energy efficiency and compliance with performance requirements.

For instance, communication transceivers are among the most energy-demanding components of IoT nodes [5]. Factors such as retransmissions, buffer constraints, and the absence of hardware filtering mechanisms can further increase energy consumption, elevate CPU utilization, and degrade overall system performance. Moreover, IoT devices may incorporate multiple transceivers and support bidirectional communication over channels subject to failures, while also operating under different CPU power states to enhance throughput and reduce energy expenditure. Given the strong interplay between energy consumption and system performance, combined with the inherent runtime heterogeneity of IoT devices, assessing energy–performance trade-offs before deployment is both complex and essential.

Although traditional network simulators such as NS-3 and OMNeT++ are widely adopted for IoT applications' analysis, these tools often lack capabilities for formal verification, structured model reuse, and detailed representation of internal device behavior (e.g., fine-grained CPU power states or inter-component delays). In contrast, Colored Petri Nets (CPNs) provide a mathematically rigorous and modular modeling framework that supports not only performance evaluation through simulation but also formal analysis of system behavior, concurrency, and state transitions [6]–[8]. CPNs also overcome several limitations of ordinary Petri Nets and Markov Chain (MC) models when representing complex systems [6]–[9]. In particular, hierarchical CPNs enable scalable and reusable models capable of capturing the multi-layered structure that characterizes modern IoT networks.

In this sense, this paper proposes a hierarchical Colored Petri Net model for energy consumption and performance analysis of IoT devices. By extending our previous work [8], this current study incorporates a detailed energy modeling for both CPU and transceiver components, supports heterogeneous CPU technologies, multiple transceivers per node, bidirectional communication channels subject to failures, and CPU-transceiver data transfer delays, features often overlooked in the literature, despite their impact on IoT systems. This enhanced modeling capability enables designers to evaluate energy consumption and throughput prior to deployment. To validate the proposed model, we compare its outputs

against measurements from real hardware, specifically an ARM7TDMI-based CPU[1] and an SX1278 LoRa transceiver[2]. The model achieves relative errors below 0.4% for CPU energy, below 2% for transceiver energy, and nearly zero for throughput.

The main contributions of this work are summarized as follows:

- a hierarchical CPN model that integrates CPU power modes, transceiver operations, channel failures, and CPU-transceiver transfer delays into a unified and modular formalism;
- a detailed energy modeling approach that supports heterogeneous CPU technologies and multiple transceivers per IoT node, capturing system behavior at the component level, a feature rarely addressed in the literature;
- a hardware-based evaluation to validate the proposed model and demonstrate its practical applicability.

The remainder of this paper is organized as follows. Section II reviews the related works. Section III describes the CPN model. Section IV presents the model validation. Finally, Section V concludes this paper and highlights potential future directions.

II. Related Work

Many studies have focused on modeling and evaluating the energy-performance tradeoff of Internet of Things (IoT) devices. Existing approaches include analytical techniques, stochastic models, formal methods, and high-level simulation frameworks. This section reviews the most relevant studies aligned with energy-performance-aware modeling for IoT systems, highlighting their contributions and limitations relative to our proposed hierarchical Colored Petri Net (CPN) model.

Shareef and Zhu [7] present a stochastic modeling approach using Stochastic Petri Nets (SPNs) to evaluate the energy consumption of wireless sensor networks under different operational policies. Their work captures fundamental energy-draining events such as retransmissions and idle listening, providing valuable insights into the energy constraints of low-power wireless devices. However, their approach does not consider component-level behavior, such as detailed CPU states or transceiver internals, nor does it model heterogeneous devices or complex communication scenarios.

Delgado et al. [10] focus on batteryless IoT devices and propose analytical models to evaluate energy harvesting, supercapacitor behavior, and energy-neutral operation (ENO). Their work is important for understanding intermittency, energy storage dynamics, and energy-driven failures in IoT systems. While these models accurately characterize energy availability, they do not incorporate communication processes, bidirectional channels, or performance-related metrics such as throughput.

Lages et al. [8] introduce a stochastic model to evaluate energy consumption in LPWAN (Low Power Wide-Area Network) networks, particularly LoRa-based systems. This prior work successfully captures transceiver behavior and energy usage but abstracts away CPU internals, power-state transitions, and multi-transceiver configurations. The current study extends that model by incorporating detailed CPU energy modeling, multiple power states, heterogeneous node configurations, and realistic channel failures, resulting in a more comprehensive representation of IoT node operations. The inclusion of CPU-transceiver transfer delays and bidirectional channel interactions constitutes a substantial advancement over this earlier work.

Correia et al. [11] present a stochastic framework for analyzing the energy consumption of LoRaWAN-based wireless sensor networks by deriving probability distribution functions (PDFs) for the total, minimum, and maximum energy consumed per operational cycle. Their numerical results show close agreement between the theoretical model and the LoRaWAN-SIM simulator[3]. The model also estimates minimum and maximum consumption bounds and demonstrates that adding gateways substantially reduces average energy consumption (e.g., from $152.12\,\text{mJ}$ to $113.83\,\text{mJ}$ when increasing from 1 to 15 gateways). Although this approach provides valuable statistical insight into network-wide energy behavior, it treats each node as a single aggregated entity and does not represent internal components such as CPU states, transceiver dynamics, or bidirectional channel effects.

Rahmati [12] proposes an Energy-Aware Federated Learning (EAFL) framework for 5G-enabled IoT networks that integrates adaptive client selection, quantization-aware training, and blockchain-enhanced security. The experimental evaluation demonstrates substantial gains: EAFL reduces energy consumption by 35.4% compared to traditional federated learning (TFL), decreasing total energy from 99.5 J to 64.2 J, while maintaining a high model accuracy of 91.8%. The communication overhead is reduced from 47.5 MB (TFL) to 23.1 MB, and the blockchain-based verification mechanism lowers the model-poisoning attack success rate by 72.3%. However, EAFL operates at a high level of abstraction and does not model internal device components, such as CPU states, transceiver operations, or bidirectional channel effects. It also lacks formal modeling capabilities and does not jointly characterize energy and communication performance at the component level.

In summary, existing studies advance energy modeling, federated learning optimization, and stochastic evaluation of IoT systems. However, they generally operate at a high level of abstraction and do not jointly capture CPU states, transceiver behavior, heterogeneous device technologies, and non-ideal channel conditions within a unified modeling framework. The hierarchical CPN model proposed in this work integrates these component-level elements into a single formal structure, enabling combined evaluation of energy consumption and communication performance under realistic operating conditions.

[1] https://documentation-service.arm.com/static/5e8e1323fd977155116a3129

[2] https://www.semtech.com/products/wireless-rf/lora-connect/sx1278

[3] https://github.com/deltazita/LoRaWAN-SIM

III. Proposed Model

As a base scenario, we consider a flexible IoT system comprising a gateway and multiple source nodes. Each node may employ different CPU technologies, support multiple CPU power states, feature multiple transceivers and their power states, and communicate over non-ideal bidirectional wireless channels.

To address the inherent complexity and heterogeneity of IoT systems, the model adopts a hierarchical CPN approach that enables modular analysis and the reuse of subcomponents. A network module encapsulates four main functional submodules: Transmitter, Receiver, CPU, and Channel. The first three represent features associated with IoT nodes, while the Channel submodule models the wireless communication medium. Together, these submodules describe the data flow and energy behavior of each node in the IoT system. The Network module also orchestrates actions triggered by upper-layer commands, such as CPU state transitions and packet transmission or reception, while maintaining the global system state.

CPNs offer a suitable structure for modeling and simulating concurrent and distributed systems with asynchronous and synchronous communication [13]. A CPN model uses a visual representation composed of places (circles or ellipses), transitions (rectangles), and arcs (directed arrows) to model concurrent systems. Places encode system states and may contain sets of tokens, each assigned a color indicating its data type. All tokens in a place share the same color, which may be simple (e.g., integers) or structured (e.g., composed of other colors/types). Transitions represent events that modify the system state by handling tokens based on defined firing rules. Guards are conditional expressions associated with transitions, controlling when they may be fired. Arcs connect places and transitions, specifying how tokens flow between states, while inscriptions define the number and type of tokens transferred [13]. The marking of a net, i.e., the current distribution of tokens across places, defines which transitions are enabled.

Hierarchical Coloured Petri Nets (HCPN) are an extension of CPNs that allow for the modeling of large-scale systems by organizing them into a structured, multi-level architecture. Instead of creating one massive, unreadable diagram, HCPNs use a "top-down" or "bottom-up" approach to manage complexity. The structure relies on two main mechanisms: Substitution Transitions, which are "macro" transitions that act as placeholders to a more detailed sub-module (page) containing its own places and transitions; and Fusion Places, which are places that exist on different pages but are functionally identical, i.e., any token added to a fusion place on one page instantly appears in the corresponding fusion place on all other pages, allowing for data sharing across the hierarchy. In essence, HCPNs provide modularization and abstraction, enabling researchers to hide low-level technical details while focusing on high-level system logic, which is essential for modeling multifaceted environments like IoT ecosystems.

The following subsections detail the proposed model. Section III-A introduces the Network module structure. Sections III-B, III-C, and III-D explain the transmission, channel, and reception procedures, via the respective modules. The CPU power states and transitions are addressed in Section III-E. This modular design provides a flexible framework to simulate various IoT scenarios, enabling the evaluation of trade-offs between performance and energy consumption under heterogeneous configurations. For clarity, the description adopts: an upper case and italic font for modules, submodules, color sets, and constructors; an italic font for places and transitions; and a bold font for functions.

A. The Network Module

The *NETWORK* module represents the top-level abstraction in the proposed hierarchical CPN model. It integrates the main functional submodules, *TRANSMITTER*, *CHANNEL*, *RECEIVER*, and *CPU*, and orchestrates the interaction among them based on commands received from upper layers. Fig. 1 illustrates its structure, where the substitution transitions *tTransmitter*, *tChannel*, *tRECV*, and *tCPU* represent these submodules, respectively.

The places *pCmdTX* (green), *pCmdRX* (red), and *pCmdCPU* (blue) receive tokens that encode commands from higher layers. These are typed using the color set *MSG*, a union of disjoint sets (*MSGSENDDATA*, *MSGLISTEN*, and *MSGCPU*), and are created via constructors *SENDDATA*, *SETLISTENING*, and *SETCPU*, respectively. In this respect, a *MSG*-type token created by *SENDDATA* in *pCmdTX* initiates a transmission, encoding the source and destination nodes, transceiver ID, payload length, and data content. Similarly, a *MSG*-type token defined by *SETLISTENING* in *pCmdRX* configures a node to listen for incoming packets, either for a finite or indefinite time. A *MSG*-type token built by *SETCPU* in *pCmdCPU* instructs a node to switch CPU states, specifying the desired power mode and frequency.

The place *AllNodes* is responsible for maintaining the global state of all nodes. Each token in this place adopts the color set *NETWORK*, defined as a list of elements of type *SIMPLENODE*. Each *SIMPLENODE* contains node metadata, CPU state, and a list of transceiver descriptions. The transceiver representation uses the tuple *TID* × *TSTATE* × *SENSITIVITY* × *LDISTRIBUTIONS* × *TIMEOUT* × *LDATASEND* × *LDATARECEIVED*, supporting both energy and communication behaviors, including probabilistic modeling of received signal strength.

To track node availability, tokens are placed in *pAvailNodes* (active nodes) or *pUnavailableNodes* (nodes in low-power states). Similarly, *pAvailTransceivers* holds information about transceivers ready to transmit or receive. Finally, three main places handle the packet processing: *pNewPacket*, which stores packets broadcasted to the environment; *pTransmitted*, which holds packets successfully decoded by a receiving node; and *pToUpperLayers*, which contains packets delivered to the application layer.

All packet tokens follow the color set *PACKET*, which includes fields such as sender, intended recipient, actual recip-

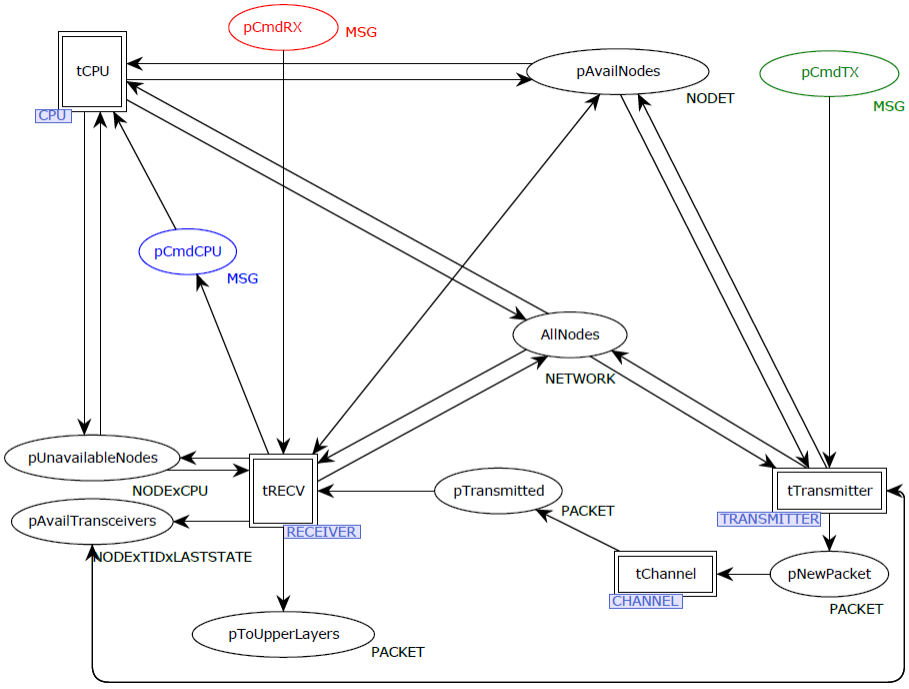

Fig. 1. The Network Module Components.

ient, packet length, transceiver ID, received power, sensitivity threshold, and payload. When a packet is received and the destination CPU is in a low-power mode, the model automatically generates a *SETCPU* command in *pCmdCPU*, emulating an external interrupt and triggering a wake-up sequence.

B. The Transmitter Module

The *TRANSMITTER* module describes the transfer of a packet from the CPU to the transceiver and into the wireless medium, as illustrated in Figure 2. A packet transmission begins when transition *tSetTransmitter* fires, which occurs when: (i) a token defined by the constructor *SENDDATA* is placed in *controlTrans*, denoting the source and destination nodes, transceiver ID, packet length, and optional payload; (ii) the source node is available in *pAvailableNode*; (iii) the corresponding transceiver of the source node is present in *pAvailTransceivers*; (iv) the guard expression *ndsrc = nodegeneral* ("node source" is equals to "general node") is evaluated to true, ensuring that the command referes to a valid node; or (v) the current network state is provided in *AllNodes*.

Firing the transition *tSetTransmitter* updates the system as follows. The transceiver's state is set to *STANDBY_TXRX* via the **updateTransceiverState** function; tokens are removed from *pAvailableNode* and *pAvailTransceivers*; a token (packet) is created in *pTransmittingToTransceiver*, delayed by a deterministic or stochastic value defined by **genTimeCpuToTrans**.

The data transfer from the CPU to the transceiver is modeled by the transition *tFromCPUtoTrans*, which fires when there is a token in *pTransmittingToTransceiver* and the network state is available in *AllNodes*. Once fired, it consumes the token from *pTransmittingToTransceiver*, generates a new one in *pTransmittingToEnv* with a timestamp produced by **genTimeTransToEnv**, representing the packet propagation delay,

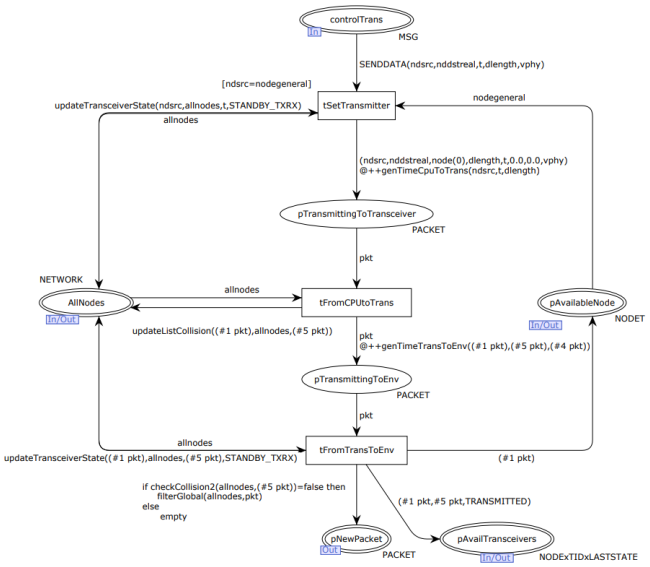

Fig. 2. The Transmitter Module.

and updates the network state to account for potential packet collisions (i.e., multiple simultaneous transmissions occurring on the same channel or interface).

The final step is modeled by the transition *tFromTransToEnv*, which fires once the packet transmission to the environment is complete. This transition requires tokens in both *pTransmittingToEnv* and *AllNodes*. Its firing yields the following effects:

- the source node is released (token added to *pAvailableNode*);
- the transceiver becomes available again (token added to

pAvailTransceivers);

- and the transceiver state is set back to *STANDBY_TXRX*.

In addition, if a collision is detected, the packet is discarded. Otherwise, all listening nodes on the same interface receive the packet, and corresponding tokens are inserted in *pNewPacket*. The function **filterGlobal** determines the received power at each destination node and filters transmissions accordingly.

In summary, the TRANSMITTER module encapsulates the packet generation and emission behavior at the physical layer, while ensuring that collisions and transceiver state transitions are accurately modeled in accordance with realistic IoT device behavior.

C. The Channel Module

The *CHANNEL* module models the effect of wireless propagation by filtering packets according to the received signal power at the destination and the receiver sensitivity threshold. It is illustrated in Fig. 3 and operates as an intermediary between the environment and the receiver module, discarding packets that do not meet the minimum signal requirements.

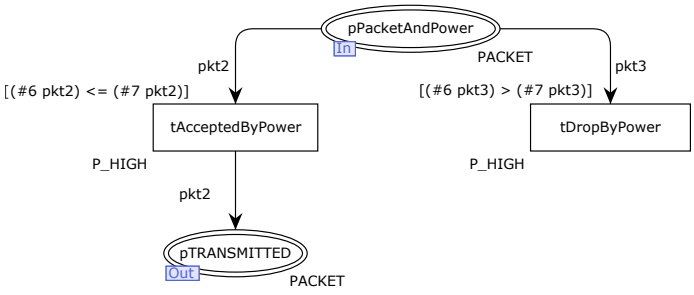

Fig. 3. The Channel Module.

Once the *TRANSMITTER* module sends a packet to the environment, a set of tokens is generated in the place *pPacketAndPower*. Each token denotes a received signal instance and carries metadata such as the received power and the transceiver sensitivity. The filtering process is governed by two transitions that consume tokens from *pPacketAndPower*, firing according to conditions that model the failure susceptibility of the wireless channel, as follows:

- *tAcceptedByPower* fires when the received signal power is greater than or equal to the destination transceiver sensitivity threshold, indicating a successful reception at the physical layer;
- *tDropByPower* fires otherwise, modeling a reception failure due to insufficient signal strength.

The firing of *tAcceptedByPower* consumes a token from *pPacketAndPower* and places one in *pTRANSMITTED*, making it available for further processing by the receiver module. When *tDropByPower* fires, the token is simply discarded, modeling a failed transmission. The *CHANNEL* module abstracts low-level channel effects such as signal degradation, reception threshold filtering, and basic packet loss. It complements the collision model handled in the *TRANSMITTER* module, enabling more realistic and granular assessment of packet delivery success.

D. The Receiver Module

The *RECEIVER* module, illustrated in Fig. 4, models the behavior of a transceiver configured to receive incoming packets and handle the reception process under different CPU power states. Specifically, it accounts for scenarios in which the CPU operates either in a high-power active state or in low-power states. The module supports both limited and unlimited listening durations and explicitly models the delay associated with transferring received data from the transceiver to the CPU, capturing the impact of reception and data handling on system performance and energy consumption.

The reception cycle starts with the transition tSetRecv, which places an available transceiver into listening mode, considering that: (1) a token defined by the constructor*SETLISTENING* exists in *pCmdRX*, specifying the node, transceiver, and listening duration type; (2) the node is available in *pAvailNodes*; (3) the global network state is present *AllNodes*.

Upon firing *tSetRecv*, the function **updateTransceiverState** sets the node's transceiver to the listening mode, and tokens are moved accordingly: the node is released back to *pAvailNodes*, and a new token is placed in *pListeningChannel*. Three distinct events may occur while a transceiver is in listening mode. The first is listening timeout (i): if the timeout expires without packet reception, the transition *tRecvTimeout* fires, consuming tokens from *pListeningChannel* and *AllNodes*, updating the transceiver state to *STANDBY_TXRX*, and releasing the transceiver to *pAvailableTransceiver*.

The second is packet received with the CPU in high-power state (ii): this scenario is handled by the transition *tAvailableNode*, which fires when the receiving node is in *pAvailNodes*, a token exists in *pTRANSMITTED* (representing a successfully transmitted packet), the transceiver is in listening mode (token in *pListeningChannel*), and the guard conditions are satisfied with the global clock greater than or equal to the timestamp on the listening token. To ensure readiness, the token timestamp may be adjusted via the function **funcTimeout**, which adds a delay margin when needed.

The firing of *tAvailableNode* consumes tokens from *pAvailNodes*, *pListeningChannel*, *pTRANSMITTED*, and *AllNodes*, updates the transceiver state to *STANDBY_TXRX*, releases the node and transceiver to their respective places, and inserts the received packet into *pToUpperLayers*, incorporating the delay for transferring the packet from transceiver to CPU.

The third and last event (iii) is packet received with the CPU in low-power state: this case is modeled by transition *tUnavailableRecv*, enabled when the destination node is in a low-power state (*pUnavailableNodes*), the transceiver is listening (*pListeningChannel*), a packet has arrived (*pTRANSMITTED*), and the global state is available in *AllNodes*. When fired, *tUnavailableRecv* simulates an external interrupt generated by the transceiver to wake up the CPU, creating tokens in *pSendWakeUp* (wake-up command), *pLowPowerNodes* and *pNodeWaiting* (intermediate wake-up state), *pNodeWaitingPacket* (queueing the received packet), and *AllNodes* (with

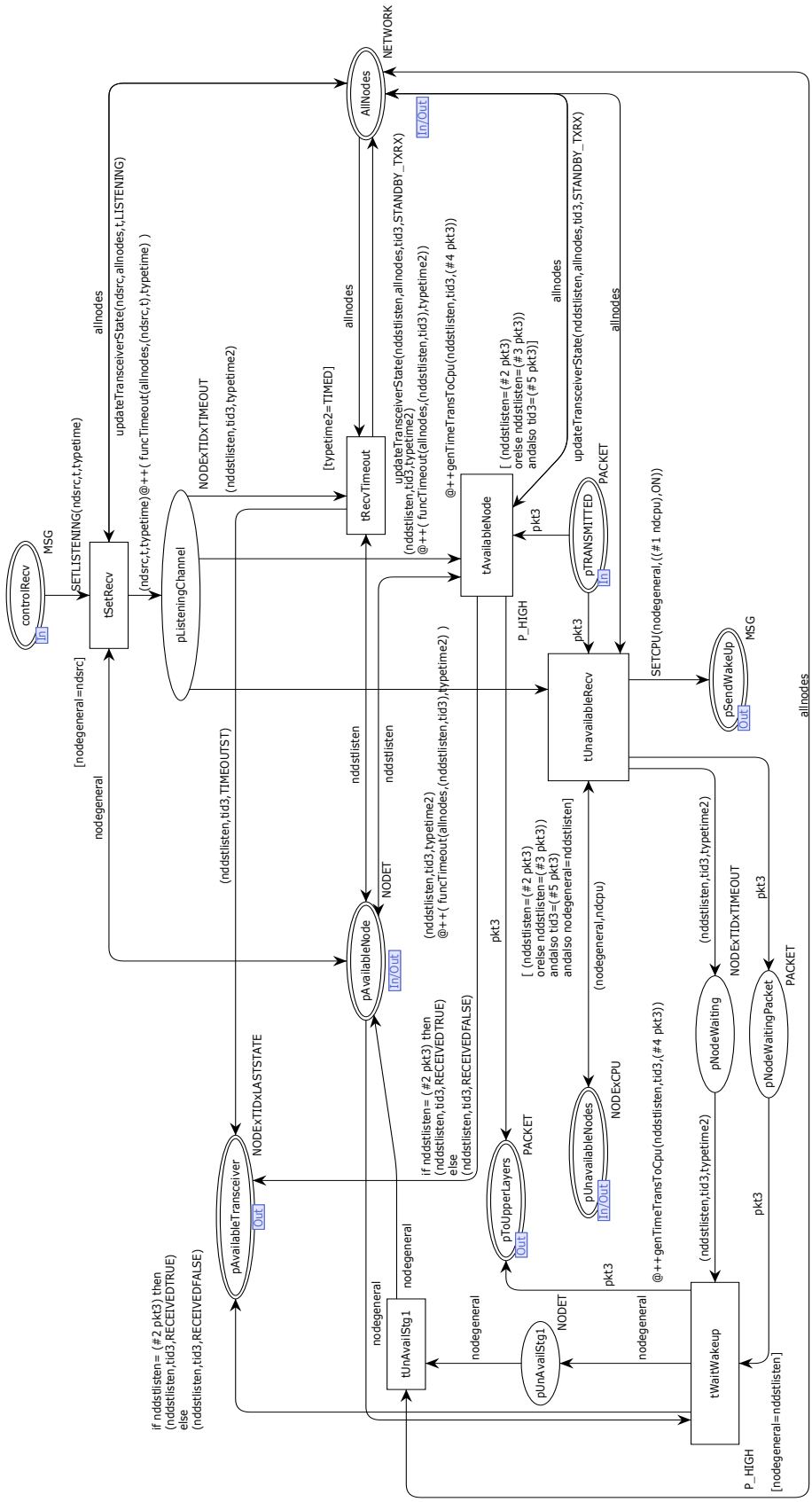

Fig. 4. The Receiver Module.

updated transceiver state).

The wake-up process unfolds in two additional steps:

- transition *tWaitWakeup* is fired once the node reaches an active state (token in *pAvailNodes*), transferring the packet to *pToUpperLayers* with the corresponding delay;
- after the transfer, *tUnAvailStg1* completes the process by updating the node's status, releasing it to the system.

This module ensures accurate modeling of real-world scenarios, including energy-efficient behavior under duty-cycled or low-power operation, which is critical for IoT deployments.

E. CPU module

The *CPU* module models the power management behavior of each node's CPU, including transitions among different power states and their associated latency overheads. It plays a central role in capturing the trade-off between energy efficiency and computational availability in the system. The model considers two main CPU power domains: a high-power *ON* state, in which the CPU is fully operational, and multiple low-power states (*SLEEP*, *STANDBY*, *IDLE*, and *POWER DOWN*) that reduce energy consumption at the expense of reduced functionality or increased wake-up latency. Transitions between these states incur non-negligible delays, representing practical overheads such as frequency scaling, context switching, or power gating. These delays are configurable for each node and CPU state and may be defined as deterministic or stochastic values derived from empirical measurements.

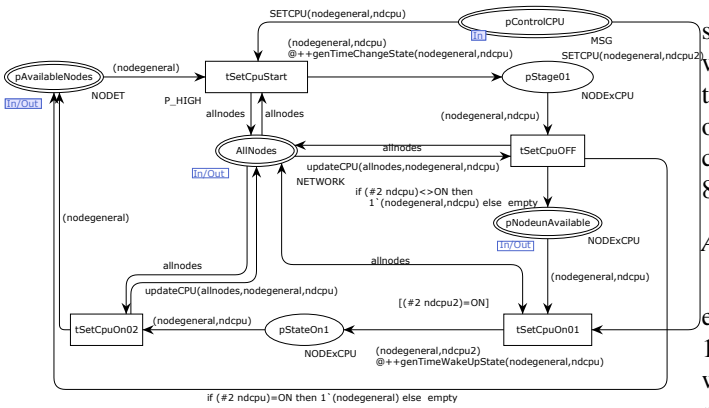

Fig. 5. The CPU Module Structure.

The state transitions are implemented as timed transitions in the model. Specifically, when a CPU state change command is issued (represented by a token with the constructor *SETCPU* in *pControlCPU*), the transition *tSetCpuStart* is enabled if the node is currently available (token in *pAvailableNodes*) and active in the network (token in *AllNodes*). Firing *tSetCpuStart* consumes the command and availableNodes tokens and places a token in *pStage01* with a timestamp delayed by the switching time, computed by the function **genTimeChangeState**. This delay represents the switching overhead, which may be deterministic or stochastic. Once this delay expires (timestamp matches the global clock), *tSetCpuOFF* fires, updating the CPU state with the **updateCPU** function. If the new state is a low-power mode, the node becomes unavailable (*pNodeunAvailable*). Otherwise, it remains available for tasks (*pAvailableNodes*).

To transition a node back to the ON state (wake-up), the *tSetCPUOn01* transition must first be enabled, which requires a `SETCPU` command token in *pControlCPU*, an unavailable node token in *pNodeunAvailable*, and a network token in *AllNodes*. When *tSetCPUOn01* fires, it removes the unavailable token and places a token in *pStateOn01*, but the CPU is not yet fully active. The token in *pStateOn01* is timestamped with a wake-up delay generated by **genTimeWakeUpState**, reflecting the time needed to resume full operation. Once this delay expires, *tSetCpuOn02* fires, consuming the *pStateOn01* token, releasing the node back to *pAvailableNodes*, and updating the CPU state to ON via **updateCPU**.

Regarding the timing functions, they are all parameterized per node and CPU power state (**genTimeChangeState**, **genTimeWakeUpState**), which allows capturing hardware-specific switching delays and supports both deterministic or stochastic modeling of these delays. This explicit modeling of timing is essential to accurately simulate the dynamic power management behavior in the system. This approach ensures that power state transitions consider switching delays and their impact on node availability, contributing to realistic energy and performance analysis of the system.

IV. Model Validation

The validation was conducted by comparing energy consumption and throughput metrics obtained from the model with those measured on real hardware under controlled conditions. To validate the model, we considered a node composed of one CPU and one transceiver. Energy consumption for both components, as well as system throughput, were analyzed over 80 transmissions, each lasting 80 seconds.

A. CPU Validation

CPU validation was carried out using five distinct nodes, each configured to operate at a different clock frequency: 12, 24, 36, 48, and 60 MHz. Only the ON power mode was used, as it enables evaluation of both CORE and IO power consumption. Tables I and II present the mean energy consumption values obtained from both the real system and the model, along with their corresponding 95% confidence intervals. The relative errors between measured and simulated values are also in the tables. The results show excellent agreement between model and real measurements, with maximum relative errors of 0.0034 for the CORE and 0.000934 for the IO power rails. These values confirm that the model accurately captures the CPU's energy behavior across a range of operating frequencies.

B. Transceiver Validation

For validating the transceiver, we configured one node to continuously transmit 255-byte packets with no idle time between transmissions, while a second node remained in

TABLE I
VALIDATION REGARDING ENERGY CONSUMPTION FOR THE CORE.

Frequency	Real (J) $\overline{x} \pm$ 95%CI	Model (J) $\overline{x} \pm$ 95%CI	Relative Error
12.0	0.8428 ± 0.000034	0.8416 ± 0.000001	0.0014
24.0	1.4444 ± 0.000027	1.4490 ± 0.000001	0.0032
36.0	2.0230 ± 0.000064	2.0161 ± 0.000002	0.0034
48.0	2.5476 ± 0.000093	2.5522 ± 0.000003	0.0018
60.0	3.0675 ± 0.000103	3.0669 ± 0.000004	0.0004

TABLE II
VALIDATION REGARDING ENERGY CONSUMPTION FOR THE IO.

Frequency	Real (J) $\overline{x} \pm$ 95%CI	Model (J) $\overline{x} \pm$ 95%CI	Relative Error
12.0	0.5545 ± 0.001974	0.5544 ± 0.000064	0.000143
24.0	0.5265 ± 0.004861	0.5264 ± 0.000181	0.000207
36.0	0.4916 ± 0.009903	0.4916 ± 0.000258	0.000067
48.0	0.4301 ± 0.012831	0.4305 ± 0.000337	0.000934
60.0	0.5222 ± 0.012506	0.5223 ± 0.000430	0.000179

continuous reception mode. Table III presents the mean energy consumption for each transceiver state (transmission, reception), comparing real measurements with model values under a 95% confidence interval. The results indicate that the model reliably estimates transceiver energy consumption, with maximum relative errors of 0.0016 for transmission and 0.0155 for reception modes. These low error margins support the use of the proposed model for evaluating energy usage in LoRa-based communication systems.

C. Throughput Validation

We also validated the model's ability to estimate throughput by comparing the number of transmitted packets over 80 seconds in both the real and simulated environments. As shown in Table IV, the model matches the actual network behavior, achieving a near-zero relative error of 8.52×10^{-11}. This result confirms the model's precision in throughput estimation for single-node transmission scenarios.

V. CONCLUSIONS AND FUTURE DIRECTIONS

This work presented a hierarchical Colored Petri Net (CPN) model for analyzing energy consumption and performance in IoT networks. The proposed model supports heterogeneous CPU architectures, multiple transceivers per node, bidirectional communication, and faulty wireless channels, offering a comprehensive representation of hardware components and communication interactions. We validated the model considering real hardware (ARM7TDMI CPU and SX1278 LoRa transceiver). The results show the model's accuracy achieved relative errors below 0.4% for CPU energy, below 2% for transceiver energy, and nearly zero for throughput estimation.

Future work includes the incorporation of multiple gateways and routing protocols, enabling the analysis of diverse network topologies and multi-hop communication scenarios. Another direction is the support for node mobility and dynamic channel conditions, which would enhance model realism by capturing time-varying deployment environments. Additionally, integrating machine learning techniques for adaptive energy optimization represents a key research avenue, allowing the system to autonomously adjust its operation to maintain energy efficiency without compromising performance. Overall, this work provides a solid foundation for energy-aware modeling of IoT systems and opens new opportunities for further advances in performance optimization and pre-deployment evaluation tools. The proposed models can also be used to perform model checking considering properties and desired behaviors of different scenarios.

TABLE III
VALIDATION REGARDING TRANSCEIVER'S ENERGY CONSUMPTION.

Mode	Real (J) $\overline{x} \pm$ 95%CI	Model (J) $\overline{x} \pm$ 95%CI	Relative Error
Transmission	3.1229 0.000779	3.1274 0.000045	0.0016
Reception	3.0146 0.005585	3.0622 0.001182	0.0155

TABLE IV
VALIDATION REGARDING THROUGHPUT.

Real (J) $\overline{x} \pm$ 95%CI	Model (J) $\overline{x} \pm$ 95%CI	Relative Error
0.23422208	0.23422208	≈ 0.0

ACKNOWLEDGMENT

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Code 001 and by the UFPE/ Propesqi via Edital Nº 06/2024

REFERENCES

[1] H. Fattah, *5G LTE Narrowband Internet of Things (NBIoT)*. CRC Press/Taylor & Francis Group, 2018.

[2] P. Ray, "A survey on Internet of Things architectures," *Journal of King Saud University - Computer and Information Sciences*, 2018.

[3] S. Sinha, "Wireless IoT Connectivity Chipset Market Report 2025-2030," IoT analytics, Tech. Rep., 08 2025. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[4] T. Korkmaz and K. Sarac, "Characterizing link and path reliability in large-scale wireless sensor networks," in *IEEE Intl. Conf. on Wireless and Mobile Computing, Networking and Communications*, 2010.

[5] J. Kaur and S. Reddy, "Operating systems for low-end smart devices: a survey and a proposed solution framework," *International Journal of Information Technology*, vol. 10, 10 2017.

[6] A. Gosavi, *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*, ser. Operations Research/Computer Science Interfaces Series. Springer US, 2014.

[7] A. Shareef and Y. Zhu, "Effective stochastic modeling of energy-constrained wireless sensor networks," *Journal of Computer Networks and Communications*, vol. 2012, no. 1, p. 870281, 2012.

[8] D. Lages, E. Borba, J. Araujo, E. Tavares, and E. Sousa, "Energy Consumption Evaluation of LPWAN: A Stochastic Modeling Approach for IoT Systems," in *IEEE Intl. Systems Conference (SysCon)*, 2021.

[9] M. Guizani, A. Rayes, B. Khan, and A. Al-Fuqaha, *Network Modeling and Simulation: A Practical Perspective*. Wiley, 2010.

[10] C. Delgado, J. M. Sanz, C. Blondia, and J. Famaey, "Batteryless lorawan communications using energy harvesting: Modeling and characterization," *IEEE Internet of Things Journal*, 2020.

[11] F. Correia, M. Alencar, and K. Assis, "Stochastic modeling and analysis of the energy consumption of wireless sensor networks," *IEEE Latin America Transactions*, vol. 21, no. 3, pp. 434–440, 2023.

[12] M. Rahmati, "Energy-aware federated learning for secure edge computing in 5g-enabled iot networks," *Journal of Electrical Systems and Information Technology*, vol. 12, no. 1, p. 13, 2025.

[13] A. Shahidinejad, M. Ghobaei-Arani, and L. Esmaeili, "An elastic controller using colored petri nets in cloud computing environment," *Cluster Computing*, vol. 23, no. 2, pp. 1045–1071, 2020.