Catching the LoRa ADR Bandit with a New Sheriff: J-LoRaNeS

Jules Courjault, Baptiste Vrigneau, Olivier Berder, Yvon Legoff, Claude Guichaoua

Catching the LoRa ADR Bandit with a New Sheriff: J-LoRaNeS

Jules Courjault
Univ. Rennes, CNRS, IRISA
Lannion, France
jules.courjault@irisa.fr

Baptiste Vrigneau
Univ. Rennes, CNRS, IRISA
Lannion, France
baptiste.Vrigneau@irisa.fr

Olivier Berder
Univ. Rennes, CNRS, IRISA
Lannion, France
olivier.berder@irisa.fr

Yvon Legoff
CG-Wireless
Lannion, France
yvon.legoff@cgwi.fr

Claude Guichaoua
CG-Wireless
Plogastel St Germain, France
claude.guichaoua@cgwi.fr

ABSTRACT

Low Power Wide Area Networks (LPWAN) are very promising for a variety of IoT applications, but they face two major challenges: energy consumption of the wireless nodes and congestion of the networks due to the huge number of nodes involved. LPWAN transmission parameters can be optimised, e.g. using artificial intelligence algorithms, but the performance estimation made during simulations is often higher than what it is in reality. In this paper, we propose a new LoRa network simulator, J-LoRaNeS. Based on the Julia programming language, it allows fast prototyping and is therefore suited to the study of different adaptive LoRa mechanisms. We used our novel simulator to clearly show the benefits of using multi-arm bandits for adaptation, but we also show that the benefits reported in the literature are not attainable when realistic network conditions are simulated.

CCS CONCEPTS

• **Computing methodologies** → **Simulation tools**; *Reinforcement learning*.

KEYWORDS

Reinforcement Learning, Multi-Armed Bandit, Network Simulator, LoRa

ACM Reference Format:
Jules Courjault, Baptiste Vrigneau, Olivier Berder, Yvon Legoff, and Claude Guichaoua. 2023. Catching the LoRa ADR Bandit with a New Sheriff: J-LoRaNeS. In *Proceedings of the Int'l ACM Symposium on Mobility Management and Wireless Access (MobiWac '23), October 30-November 3 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3616390.3618279

1 INTRODUCTION

Several long-range wireless technologies, grouped under the term LPWAN (Low-Power Wide Area Network), have appeared in recent years, with the aim to fill the growing need to gather information from various connected objects. The Long Range (LoRa) [19] technology, supported by Semtech and the LoRa alliance, offers a very good trade-off between range, data rate and energy consumption. It allows a throughput ranging from 0.3 kbps to 50 kbps in the 433 MHz, 868 MHz or 915 MHz Industrial, Scientific and Medical (ISM) bands using the principle of chirp spread spectrum. The main parameters of the communication are the Spreading Factor (*SF*), Transmit Power (*TP*), bandwidth and coding rate. The throughput, energy consumption and transmission performance thus vary according to the choice of these parameters [7].

To cope with a possible huge number of nodes and ensure the reception of transmitted data over varying propagation channels, a mechanism for adapting the communication parameters called Adaptive Data Rate (ADR) is used [3]. This algorithm suffers from several problems [12], in particular a long convergence time, leading to a significant increase of power consumption at network start-up and topology changes. Proposals have been made in the literature to improve the performance of the ADR, some of them [23, 24] using Multi-Arm Bandit (MAB), a reinforcement learning technique. Those algorithms assume full and perfect feedback, where duty cycle constraints are not taken into account, which leads to an overestimation of the performance gain.

A network simulator is very useful before deployment to test multiple available configurations, e.g. which ADR suits best your network needs. Several LoRa network simulators have arisen in recent years [2, 4, 13, 14, 21, 25]. But none of them can fully claim that it provides rapid simulation and user-friendly customization. Recently developed by the MIT, Julia [1] represents a good trade-off between low or intermediate level languages such as C/C++ and high-level languages such as Matlab or Python. The contributions of the present paper are:

- The design of a new simulator based on Julia, the exploration of its architecture and the proposed metrics.
- The comparison of MAB-based ADR to LoRaWAN ADR with and without perfect feedback.

The paper is organized as follows: Section 2 presents the state of the art on LoRa network simulators, as well as the architecture of J-LoRaNeS. Section 3 gives details on the different ADR that will be studied in this paper, especially those based on MAB. Finally Section 4 presents simulations results and comparisons of ideal and realistic schemes in terms of packet delivery ratio and energy, while conclusions are drawn in Section 5.

MobiWac '23, October 30-November 3 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0367-6/23/10…$15.00
https://doi.org/10.1145/3616390.3618279

Table 1: Comparison of LoRa networks simulators.

Simulator name	ISFO	Energy	ADR	Smart node
COOJA [17]	×	✓	✓	✓
TOSSIM [11]	×	✓	✓	✓
CupCarbon [5, 15]	×	✓	✓	✓
LoRaWANSim [14]	✓	✓	Static	×
LoRaSim (Java) [25]	×	✓	✓	×
LoRaSim (Python) [4]	×	✓	×	×
LoRa-MAB [23]	✓	✓	Custom	✓
LoRaFREE [2]	✓	✓	Custom	×
FLoRa [21]	×	✓	✓	×
LoRaWAN ns-3 [13]	✓	✓	✓	×
J-LoRaNeS	✓	✓	✓	✓

"Static": The ADR algorithm is used once and not re-applied.
"Smart node": The simulator can run machine learning algorithm.

2 J-LORANES: A JULIA BASED LORA NETWORK SIMULATOR

2.1 Why a new simulator?

A network simulator can be very useful to test a network configuration, e.g. compare different ADR algorithms before in-field deployment, and potentially save significant resources by revealing poor performance at simulation. Several simulators have been proposed in scientific literature using different languages and offering different abilities and results. Most of them are listed in Table 1, comparing their respective pros and cons. Four simulation capabilities are particularly considered, i.e., the imperfect *SF* Orthogonality (ISFO), energy consumed, ADR nature and smart node. ISFO prevents two packets emitted simultaneously, with different *SF*, to be received. Thus not considering ISFO in simulation leads to an overestimation of the network performance (one should note that under certain conditions one of the packets can be received thanks to the capture effect). ADR nature refers to the kind of ADR mechanism used in each simulator, either LoRaWAN ADR, MAB-based or custom ADR. The smart node capability corresponds to the capacity of making a simulation with reinforcement learning based ADR on nodes.

One of the first LoRa network simulator proposal was LoRaSim (Python) [4], a discrete event simulator based on SymPy. It was designed to study LoRa scalability with network energy consumption and performance monitoring, but is limited since it does not take into account the ISFO, downlink traffic and ADR mechanism. Since it is an open-source simulator, it has been extended to add many missing features while testing new mechanisms to enhance the performance of the ADR algorithm [2, 8].

LoRa-MAB [23] is also based on SymPy and was developed to study the performance of a rate adaptation mechanism based on MAB. This simulator takes into account ISFO, but the ADR of LoRaWAN is not available. Another simulator called LoRaSim [25] was developed using the Java language. It aims to simulate LoRaWAN networks in the US 915MHz band, and is more complete than the LoRaSim (Python) simulator, but ISFO is still missing.

COOJA [17], TOSSIM [11] and CupCarbon [5, 15] are simulators dedicated to LPWAN technologies and not specifically to LoRa. COOJA and TOSSIM are able to emulate code that will be deployed on End-Devices (ED), while CupCarbon needs the ED behavior to be described in SenScript.

LoRaWANSim [14] is a simulator based on Matlab. The goal of this simulator is to provide a tool as accurate as possible for LoRaWAN network, thus ISFO and ADR are available. However, the ADR algorithm used in this simulator is static: for each pair of gateway (GW) and ED, 20 SNR levels are randomly generated with respect to the propagation model. Subsequently, a value of *SF* and *TP* is assigned to each ED thanks to the ADR algorithm presented in section 3.1. This *SF* and *TP* value selection is made only once during the whole simulation.

FLoRa [21] and LoRaWAN ns-3 [13] are coded with the C++ programming language, they need OMNeT++ and ns-3 respectively, which can make them hard to use for non-Linux users. Both of the simulators feature the ADR algorithm, but only LoRaWAN ns-3 takes into account ISFO.

Our simulator, J-LoRaNeS, is based on the Julia programming language, which is a compiled language, yet allowing high-level syntax, offering a good trade-off between ease of development and fast execution time. Moreover, it has been developed to be as accurate as possible, therefore ISFO, bidirectional traffic and dynamic ADR are implemented. Thanks to the multiple-dispatch feature of Julia, it is straightforward to replace several parts of the simulator, such as the ADR algorithm, without affecting the core algorithm of J-LoRaNeS.

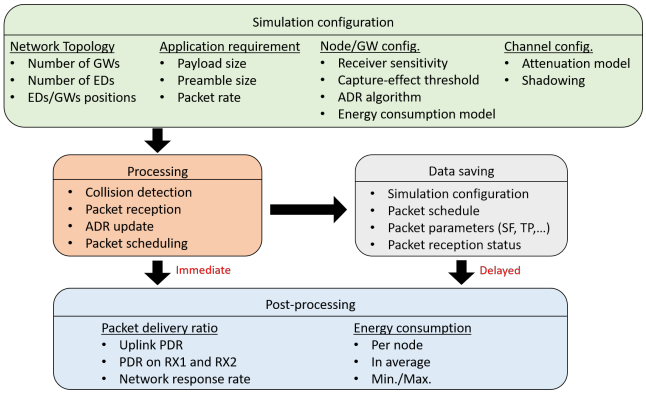


Figure 1: Simulator configuration and results.

2.2 J-LoRaNeS architecture

Figure 1 presents an overview of J-LoRaNeS. First the user has to describe the topology of the network, the application requirements (packet rate, payload size, preamble size...), the hardware configuration (sensitivity, antenna gain, energy consumption model,...) and the channel used for the simulation. The simulation takes place in the processing step: the packet schedule is generated, collisions between packets are detected, reception of packets is checked and the ADR is updated. When the processing step is complete, data are saved for post-processing. After the saving step, the user can apply different built-in metrics to monitor the performance of the network, e.g. Packet Delivery Rate (PDR) or energy consumption.

2.2.1 Packet scheduler. The aim of the scheduler is to decide when to transmit the different packets while respecting the Duty Cycle (DC) and packet rate constraints. There are two types of packet to manage: the uplink (UL) packet from the ED and the downlink (DL) one from the GW.

Table 2: SNR_{min} value in dB used to compute sensitivity threshold [20]

SF	7	8	9	10	11	12
SNR_{min}	-7.5	-10	-12.5	-15	-17.5	-20

Table 3: Collision threshold in dB to exceed in order to receive one of the colliding packets thanks to capture effect [9]

	*SF*7	*SF*8	*SF*9	*SF*10	*SF*11	*SF*12
*SF*7	6	-16	-18	-19	-19	-20
*SF*8	-24	6	-20	-22	-22	-22
*SF*9	-27	-27	6	-23	-23	-25
*SF*10	-30	-30	-30	6	-26	-28
*SF*11	-33	-33	-33	-33	6	-29
*SF*12	-36	-36	-36	-36	-36	6

DL packets are scheduled after an UL is received. There are two possible windows to schedule them: RX1 or RX2. Those windows have different parameters (*SF*, frequency, DC). On RX1, a DL packet uses the same parameters as the corresponding UL packet, whereas on RX2 it uses a predefined set of parameters, usually $SF = 12$, with maximum TP, but with different central frequency from the UL. An example of frame scheduling is given on Figure 2. If multiple GWs are available in the network (i.e. multiple GW have available DC), the one that received the UL packet with the highest Received Signal Strength Indicator (RSSI) will be selected to send the DL packet.

UL packets are dynamically scheduled based on the ALOHA protocol and the average packet periodicity of the node with respect to the duty cycle. During the initialization of the simulation, only one UL packet is scheduled for each node. The others are scheduled after the reception of an UL packet, if it has not generated any DL packets, or after the processing of the scheduled DL packet.

2.2.2 *Packet reception.*

Channel effect. To be successfully decoded, a packet must be received with an RSSI greater than a sensitivity value depending on the *SF* value of the packet [20]. The channel model is described in the channel configuration file, and can be easily modified (with distance or random variables). The sensitivity threshold S is computed as follows:

$$S = -174 + 10\log_{10}(BW) + NF + SNR_{\text{min}}, \quad (1)$$

where the first term corresponds to the thermal noise, NF is the noise factor of the radio which is hardware dependent, and SNR_{min} is the minimum SNR required to receive a packet at a given SF, the values of SNR_{min} are given in Table 2 for each SF.

Interference effect. Due to the ISFO of LoRa, packets sent on different *SF* might interfere between them, but one of the packet might be retrieved thanks to the capture effect. Indeed, if the Signal to Interference plus Noise Ratio (SINR) is higher than a threshold, given in Table 3, the packet can be decoded [9].

2.2.3 *Customising the ADR algorithm.*

To test or customize the ADR in the simulator, one must describe several functions: `reward`, `updateADRpolicy` and `selectDR`. The `reward` function will output the parameters needed for the ADR update, e.g. the reward for MAB-based ADR or the RSSI for the server-side algorithm of the LoRaWAN ADR. The `updateADRpolicy` function defines how the ADR policy is updated and `selectDR` selects the new communication parameters according to the updated policy. The last two functions profit from the multiple dispatch of Julia, *i.e.* each function is defined multiple times to yield a specific behavior according to the type of the packet. A packet can be of types `UL`, `DLRX1` or `DLRX2`, which correspond to packets sent as UL, DL on RX1 window and DL on RX2 window, respectively. If a packet of type `UL` is processed the `updateADRpolicy` function will need to update the policy on the server side. Whereas, for `DLRX1` or `DLRX2` it needs to update the policy on the ED side. The dispatch is the same for the function `selectDR`, *i.e.*, the communications parameters will be selected according to the policy and method of the server if an `UL` packet is received, and according to the ED policy and method for types `DLRX1` or `DLRX2`.

Algorithm 1 J-LoRaNES algorithm

```
EDs, GWs ← DevicePositioning(NED, NGW)
channelEffects ← ComputeChannel(EDs, GWs, CModel)
packetList ← ScheduleULpackets(EDs, timeOffset)
packetKey ← GenerateKey(packetList)
m ← 1
while m < size(packetKey) do
    packet ← GetPacket(packetList, packetKey[m])
    rssiLvl ← RSSI(packet, channelEffect)
    receivedNoInterference ← rssiLvl > Sensitivity
    interfererList ← FindInterferer(packet, packetList)
    if Isempty(interfererList) then
        received ← receivedNoInterference
    else
        received ← CompareRSSI(packet, interfererList) and receivedNoInterference
    end if
    dlScheduled ← ScheduleDL(packet, rssiLvl, packetKey, packetList) {MD}
    UpdateADRpolicy(packet, received, rssiLvl) {MD}
    SelectDR(packet, received, dlScheduled){MD}
    ScheduleUL(packet, received, dlScheduled, packetKey, packetList){MD}
    m ← m + 1
end while
```

"MD": The function uses multiple-dispatch

2.2.4 *Core of the simulator.*

Algorithm 1 presents the simulator algorithm, and how the elements presented above are organized. The network is first initialized according to the inputs given by the user, including the propagation model and device positioning. Then, the simulator schedules one UL for each ED and generates the variable vector `packetKey`, which allows the identification of a packet. Once the initialization step is complete, the algorithm performs, for each packet key, the following operations: *i)* Find the packet associated to the key; *ii)* Check if the packet RSSI is above or below the sensitivity threshold for reception; *iii)* Find all packets

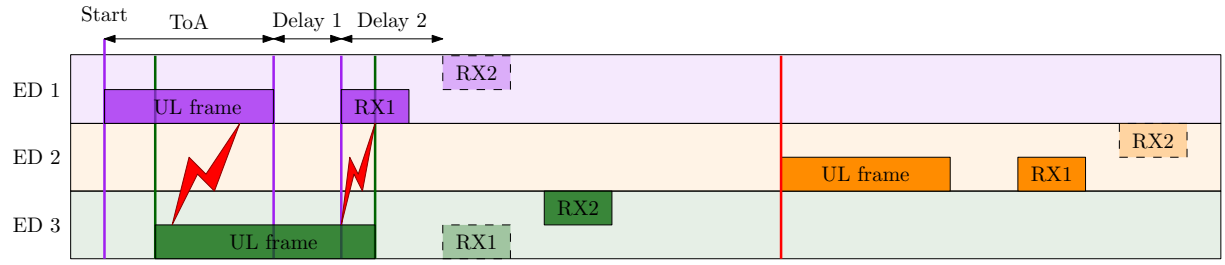


Figure 2: Frames in time and frequency: cases with no and interference.

that interfere with the one being processed and compare the SINR to a threshold value defined by Table 3 to decide if the packet is delivered or not.

Functions `ScheduleDL` and `ScheduleUL` are used to schedule DL and UL respectively. Both of them are using multiple-dispatch, and it is made on the type of the packet. If the packet is of type `UL`, the scheduling is made as described in paragraph 2.2.1. Whereas, for `DLRX1` and `DLRX2` packet, the `ScheduleDL` function will do nothing, and an UL will be scheduled by the `ScheduleUL` function.

This solution offers the possibility to easily add new node profiles or change some algorithms without changing the core algorithms of J-LoRaNeS. For example, if the processed packet is a correctly received UL and considering the LoRaWAN ADR described in 3.1, then a DL packet is scheduled on RX1 or RX2 according to the availability of the GW, and ADR is updated. For each received UL packet, the corresponding SNR value is computed and stored for later use. In case of no scheduled DL packet or unsuccessful DL transmission, the failed transmission counter is increased on node side of the ADR. The ADR chooses a new data rate, according to the selected method, in this example the LoRaWAN ADR. When the new communication parameters have been chosen, an UL is scheduled for the device. This new packet is added to the `packetList` variable, if the end of the transmission does not exceed the simulation time. A new UL packet is scheduled if the processed packet is a DL, or `dlScheduled` is false for UL packet.

3 ENHANCING THE ADR WITH MAB

To demonstrate the capabilities of J-LoRaNeS, we propose a study of the ADR proposed for LoRa network. As explained in Section 1, the LoRaWAN ADR [3] suffers from a long convergence time, and several propositions of improvement have been made in the literature. Some of the solutions [23, 24] are based on MAB techniques, which need feedback to work properly. In those solutions it is assumed that the feedback is made through DL packets, and happens after each received UL packet. However, when transmitting on ISM bands, LoRa equipment must respect a duty cycle (DC) of 1 or 10% depending on the regions and bands used. Thus in high density networks, GWs will not be able to send a DL for each received UL, leading to an imperfect feedback. The question our study aims to answer is: **what happens to the performance of MAB-based ADR algorithms when the DC is properly respected?**

3.1 LoRaWAN ADR

The LoRaWAN ADR is described in the LoRaWAN v1.1 specification [3]. This algorithm adapts LoRa parameters to the observed channel properties, by dynamically changing SF and TP values. This algorithm is split into two parts. On every node, it evaluates the wireless communication quality by counting the received acknowledgement packets (ACK) from the GW. If the number of missing ACKs is too high, the node increases first the TP and next the SF in order to enhance the radio link. The second part of this mechanism is centralised on the network server, trying to make communication more energy efficient. To do this, it records the SNR of several (usually 20) UL packets and then computes the margin on SNR, which is used to determine how TP and SF can be modified to reduce the power consumption of the transmission, while still receiving the UL.

3.2 MAB-based ADR

With a MAB approach, the mechanism is decentralised, there is no part of the mechanism running on the network server. Figure 3 shows the basic operation of MAB algorithms applied to LoRa. First, the agent chooses an arm, i.e. a combination of SF and TP parameters for the next communication, then performs the transmission with the chosen parameters. The reception of this communication by the network will lead to the transmission of a DL packet. The reception of this DL packet by the ED leads to the reward computation, at instant t_i: $r[t_i] = 1$ if the DL is received, $r[t_i] = 0$ otherwise. The next steps are updating the policy of the algorithm, i.e. the way the arm are chosen, then choosing the communication parameters. Those steps are performed differently depending on the nature of the algorithm used. The MAB algorithms used in this article for parameters selection and policy update are $\epsilon-$Greedy [22] and Thomson-Sampling (TS) [18].

3.2.1 $\epsilon-$Greedy. For each arm k, a value $\hat{\theta}_k$ is associated, which corresponds to the average reward received by the arm when it was chosen. The update of the policy thus consists in computing

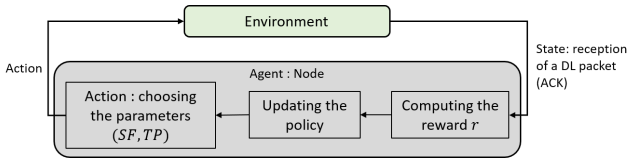


Figure 3: MAB principle applied to LoRa channel adaptation

$\hat{\theta}_k$ for the arm used once the reward has been received. For the arm selection step, the algorithm must choose between *exploiting* or *exploring*. If it chooses to exploit, then the arm with the largest $\hat{\theta}_k$ value will be selected. In case of exploration, an arm is drawn at random. The choice between exploration or exploitation is made randomly, the algorithm will explore with probability $\epsilon \in]0;1]$. In this article we have chosen $\epsilon[t_i] = \frac{N_b}{N_b + \sum_k n_k[t_i]}$, with N_b the total number of arms and $n_k[t_i]$, the number of transmissions made with the parameters of arm k. This choice of a decreasing value in time for ϵ allows a strong exploration at deployment, then a strong exploitation after a certain time, thus ensuring a use of parameters bringing a high reward.

3.2.2 Thomson-Sampling. Each arm k is associated with a random variable following a beta distribution of parameters (α_k, β_k) with $\alpha_k \geq 1$ and $\beta_k \geq 1$. The mean value of that distribution is $\hat{\theta}_k = \frac{\alpha_k}{\alpha_k + \beta_k}$. At initialization, the parameters of the beta law have the following values: $\alpha_k = \beta_k = 1$. The update of the policy consists in changing the parameters of the distribution: $(\alpha_k[t_{i+1}], \beta_k[t_{i+1}]) = (\alpha_k[t_i], \beta_k[t_i]) + (r[t_i], 1 - r[t_i])$, if the arm k has been chosen, otherwise the parameters remain unchanged. To choose the arm that will be used for the next transmission, the algorithm makes a random draw for each arm according to each beta distribution, and chooses the arm with the highest value.

Figure 4 presents an example with 3 arms, with different Beta distributions. In this example the arm 3 has the lowest average, moreover its variance is also low, thus it is not worth considering using this arm, because the received reward will most likely be lower than others arms. On the contrary the arm 1 has the highest average and also has a rather low variance, thus the reward will most likely be high. Yet, it is still worth to explore the arm 2 due to its high variance and average close to the one of arm 1, thus the probability to get a higher reward than arm 1 is significant.

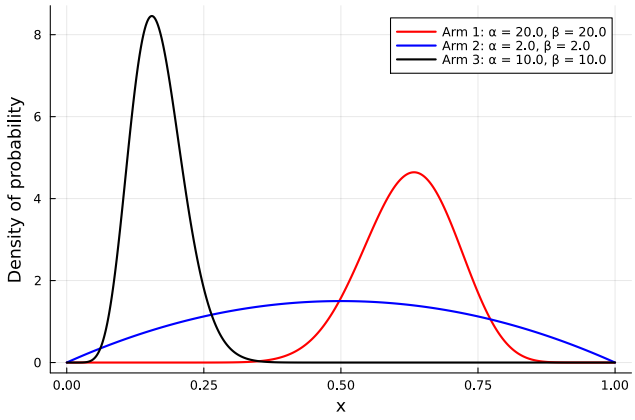


Figure 4: Probability density function of a Beta distribution in 3 different configurations

4 SIMULATIONS SETUP AND RESULTS

To estimate the real contribution of the algorithms presented above compared to the classical ADR, we use the simulator presented in Section 2. The simulated configuration is a network with one GW, 500 EDs uniformly distributed in a square of dimension 20 km, sending one packet every 10 minutes on average. The channel model used in the simulations is the Okumura-Hata [10] model for small and medium-size cities.

The different ADR mechanism presented is this paper have been implemented, and thanks to the the multiple-dispatch, it is only needed to change the type of the ADR in the initialization of the network, *e.g.*, using the keyword Greedy instead of LoRaWAN if you want to use an $\epsilon-$Greedy algorithm instead of the LoRaWAN ADR. The `reward` for the LoraWAN algorithm is the tuple (`received`, `RSSI`).

The adaptation mechanisms that will be compared are: the LoRaWAN ADR, an ADR using the $\epsilon-$Greedy MAB algorithm [22], and another using the TS algorithm [18]. The arm combinations used for MAB algorithms are identical to the possible combinations for LoRaWAN ADR, i.e. for $SF = 7$ the possible TPs range from 2 dBm to 14 dBm in 3 dB steps, and for SFs ranging from 8 to 12, only the 14 dBm TP can be selected.

In the case of the MAB ADRs, we will compare the *oracle* cases with the *DC-constrained* cases. The oracle case assumes the GW sends an ACK for each of the received UL packets, thus not respecting the DC. Note that this is the assumption made in the scientific literature [23, 24]. In the DC-constrained case the GW respects the DC of 1 or 10% imposed by regulation, in this case any UL packet received is not necessarily followed by an ACK, **resulting in a bad reward computation by MAB algorithms**.

4.1 Temporal variation of the Packet Delivery Ratio

Figure 5 shows the evolution of the PDR for the ADRs. The PDR is computed over a sliding window of one hour. In this figure, we can see that the MAB algorithms have a much lower convergence time than the LoRaWAN ADR. On one hand, the PDR obtained after convergence is better than the LoRaWAN ADR in both Oracle and DC-constrained versions. Unfortunately, it turns out that in the

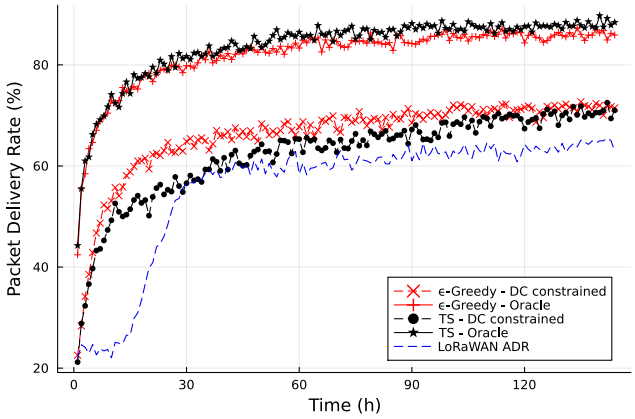


Figure 5: Packet delivery rate for $\epsilon-$Greedy and TS algorithm, in both DC-constrained and oracle cases, compared with LoRaWAN ADR.

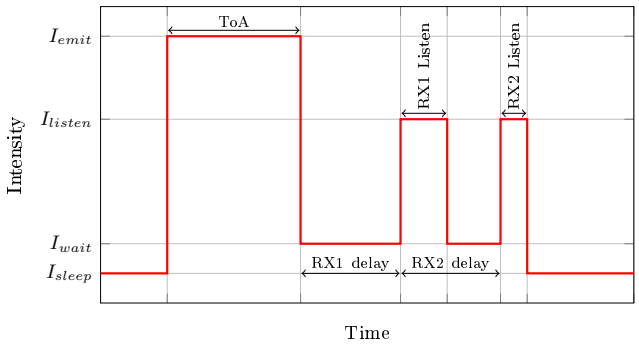


Figure 6: Energy consumption model used in the simulator

DC-constrained case, the MAB algorithms are less efficient than in the oracle cases. Indeed, in the oracle cases, both algorithms perform well, with a PDR around 85% at the end of the simulation, whereas the performance in the DC-constrained case is around 70% for the Greedy algorithm, and seems to continue to increase for the TS one. For the LoRaWAN ADR, the PDR does not go above 65%. Moreover, the convergence time is about 40h while it needs 24h for the Greedy algorithm and 48h for the TS algorithm, but the TS algorithm always has better performance than the LoRaWAN one.

This performance degradation in the DC-constrained case is due to the poor reward calculation caused by the absence of ACK, despite the reception of the UL packet, thus underestimating the performance of the arms with good performance, whereas the arms with bad performance are less impacted. We notice a decrease in the performance of the $\epsilon-$Greedy case due to the choice of ϵ, the exploration probability. Indeed, the algorithm will be stuck on a non-optimal arm, due to the low value ϵ after 30 hours of simulation. This behaviour continues until the average value of this arm falls below that of another arm. This problem could be solved by using a fixed value for ϵ, but to avoid to dramatically degrade the performance of the algorithm, the value of ϵ must be small, leading to a long convergence time. In the TS case the learning is much slower because the variance decay is also slower due to the reward miscalculation, but this allows the algorithm to be more resilient to reward miscalculations compared to the $\epsilon-$Greedy algorithm.

4.2 Topological comparisons after convergence

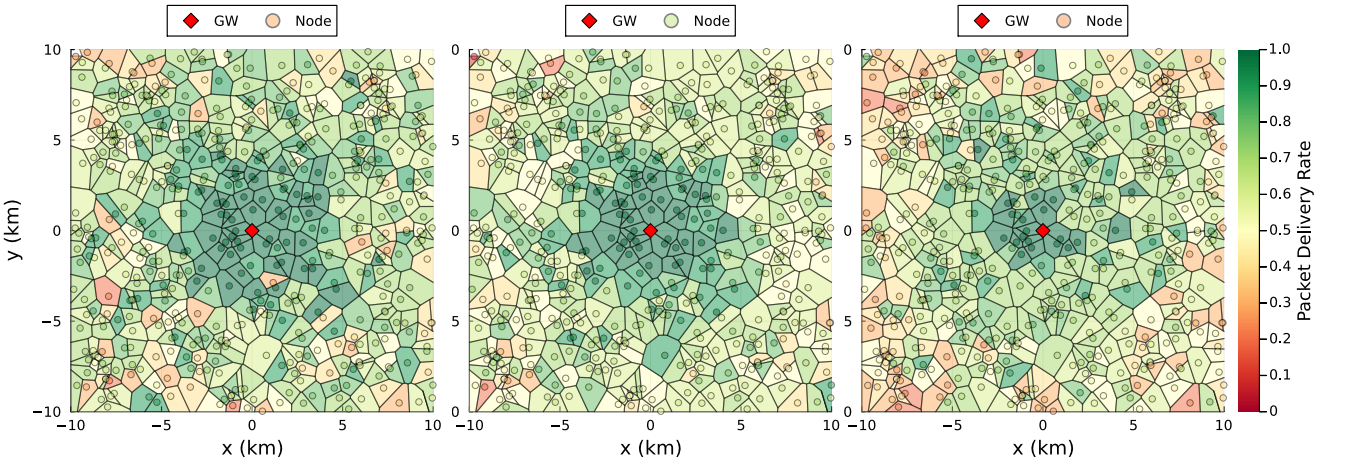
Our new simulator is capable of providing a wide variety of results. For example, Figure 7 shows the PDR averaged over the last two hours of simulation for each network node under DC-constrained conditions for the $\epsilon-$Greedy, TS and LoRaWAN ADR algorithms. We employed a Voronoi tessellation, which facilitates the visualisation of the map and its interpretation. On these maps, it can be seen that the best performing nodes are those closest to the GW, as expected. On the other hand, this representation allows a further investigation, showing that the closest nodes to the GW have acceptable performance (PDR $\geq$ 80%) within a radius of 4 to 5 km for the TS algorithm, against 3 km for the LoRaWAN ADR. It is therefore the nodes at the edge of the network that suffer from a degradation of their performance due to the lack of ACK induced by to the respect of the DC.

Figure 8 presents the most used SF over the last two hours of simulation for each network node under DC-constrained conditions for the $\epsilon-$Greedy, TS and LoRaWAN ADR algorithms. By correlating this map with Figure 7, we see that the poor performance of the MAB algorithm is due to bad arm selection. Indeed, for the $\epsilon-$Greedy algorithm, the parameter selection seems completely random.

The TS algorithm is doing better since it selects the lower SF for node close to the GW, with few exceptions. Moreover, it is able to use low SF values on far nodes, which is not the case for the LoRaWAN ADR. Figure 9 presents the energy consumption for each ED in the network for the MAB-based ADR algorithms in the DC-constrained case and the LoRaWAN ADR. The energy model used is the one presented on Figure 6 with the following parameters values [6, 16] $I_{sleep} = 1.6\ \mu A$, $I_{wait} = 27\ mA$, $I_{listen} = 38\ mA$, and $I_{transmit}$ ranging from 22.3 mA to 38 mA depending on the value of TP. As it can be seen, the results are very similar to the SF map of Figure 8, which expected. Indeed, increasing the SF value by one doubles the time on air of the transmission, thus increasing the energy consumption of the communications. Moreover, the arm configuration used in this article implies a TP of 14 dBm when the SF value is not 7.

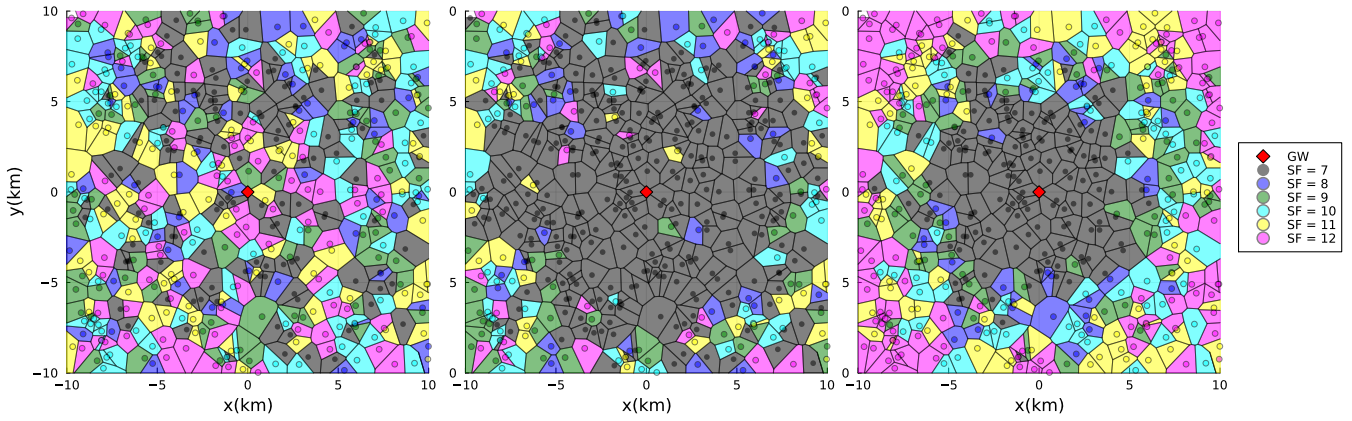
5 CONCLUSIONS AND PERSPECTIVES

In this paper we present a new LoRa network simulator based on Julia, simultaneously bringing flexibility and efficient simulation time. To demonstrate the potential of our simulator, we propose a case study on communication parameters adaptation of LoRa networks. We use our simulator to compare MAB-based ADR working with and without respecting the DC, in a network with high node density. The results show a strong degradation of the performance of the MAB algorithms in the DC-constrained case. Although the very good results presented in the literature for MAB algorithms are not achievable in practice, they are still very promising and should enhance network performance compare to LoRaWAN ADR. It is now necessary to think about a solution that could work in cases with high density of EDs in the network, especially for those further from the GW. Another important aspect that has not been addressed in this paper is the optimisation of energy consumption of the nodes. Managed in the LoRaWAN algorithm, the question of energy optimization could also be implemented by the MAB algorithms by adding a component taking into account the energy consumption in the reward calculation. The weight of this component would be more important in cases of low energy consumption, thus favouring low energy consumption arms in case of similar PDR between several arms. As demonstrated in this article, J-LoRaNeS can definitely be considered as a valuable tool to study LoRa networks performance in different situations. As an example, instead of the Gaussian environment considered in this paper, it is planned to add impulsive noise to simulate communications in industrial environment.



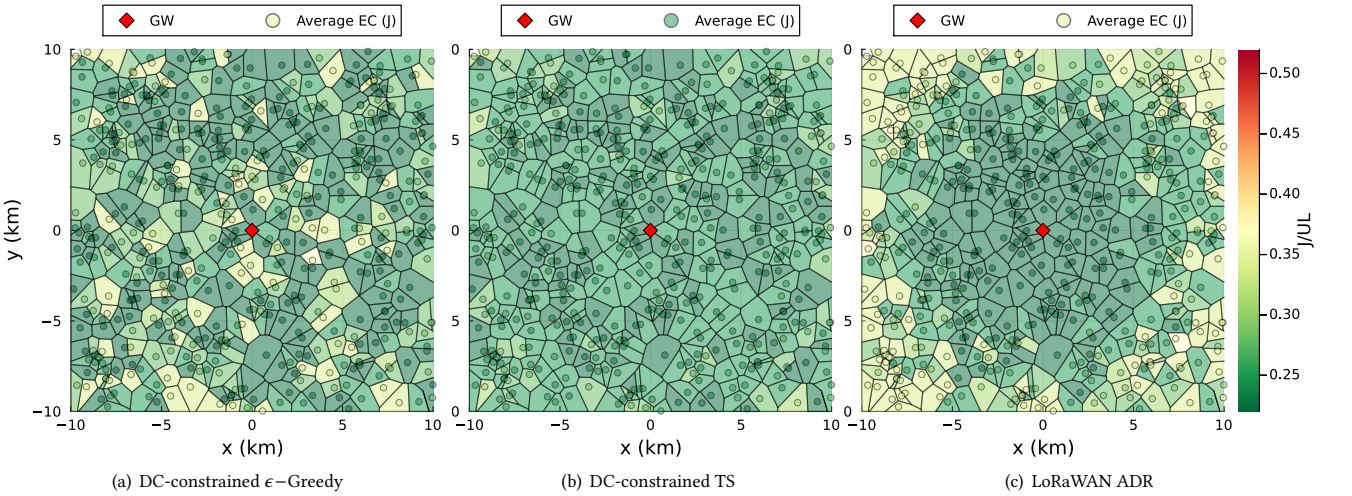
(a) DC-constrained $\epsilon-$Greedy (b) DC-constrained TS (c) LoRaWAN ADR

Figure 7: Averaged PDR on every ED of the network at the end of the simulation



(a) DC-constrained $\epsilon-$Greedy (b) DC-constrained TS (c) LoRaWAN ADR

Figure 8: Most used *SF* for every ED of the network at the end of the simulation



(a) DC-constrained $\epsilon-$Greedy (b) DC-constrained TS (c) LoRaWAN ADR

Figure 9: Averaged energy consumption for each ED of the network

REFERENCES

[1] Julia language. https://julialang.org/.

[2] K. Q. Abdelfadeel, D. Zorbas, V. Cionca, and D. Pesch. *FREE* —fine-grained scheduling for reliable and energy-efficient data collection in LoRaWAN. *IEEE Internet of Things Journal*, 7(1):669–683, 2020.

[3] L. Alliance. LoRaWAN 1.1 specifications, 2017.

[4] M. C. Bor, U. Roedig, T. Voigt, and J. M. Alonso. Do LoRa low-power wide-area networks scale? In *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM, page 59–67, New York, NY, USA, 2016.

[5] A. Bounceur, L. Clavier, P. Combeau, O. Marc, R. Vauzelle, A. Masserann, J. Soler, R. Euler, T. Alwajeeh, V. Devendra, et al. CupCarbon: a new platform for the design, simulation and 2d/3d visualization of radio propagation and interferences in IoT networks. In *15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4, 2018.

[6] L. Casals, B. Mir, R. Vidal, and C. Gomez. Modeling the energy performance of LoRaWAN. *Sensors*, 17(10):2364, 2017.

[7] J. Courjault, B. Vrigneau, O. Berder, and M. R. Bhatnagar. A computable form for lora performance estimation: Application to Ricean and Nakagami fading. *IEEE Access*, 9:81601–81611, 2021.

[8] F. Cuomo, M. Campo, A. Caponi, G. Bianchi, G. Rossini, and P. Pisani. EXPLoRa: Extending the performance of LoRa by suitable spreading factor allocations. In *IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–8, 2017.

[9] C. Goursaud and J.-M. Gorce. Dedicated networks for IoT : PHY / MAC state of the art and challenges. *EAI Endorsed Transactions on Internet of Things*, 1, Oct. 2015.

[10] E. Harinda, S. Hosseinzadeh, H. Larijani, and R. Gibson. Comparative performance analysis of empirical propagation models for lorawan 868MHz in an urban scenario. In *5th World Forum on Internet of Things (WF-IoT)*, pages 154–159. IEEE, 2019.

[11] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, 2003.

[12] S. Li, U. Raza, and A. Khan. How agile is the adaptive data rate mechanism of LoRaWAN? In *IEEE Global Communications Conference (GLOBECOM)*, pages 206–212, 2018.

[13] D. Magrin. Network level performances of a LoRa system. 2016.

[14] R. Marini, K. Mikhaylov, G. Pasolini, and C. Buratti. LoRaWANSim: A flexible simulator for lorawan networks. *Sensors*, 21(3), 2021.

[15] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. In *7th International ICST Conference on Simulation Tools and Techniques*, pages 126–131, 2014.

[16] Microchip. RN2483: Low-Power Long Range LoRa Technology Transceiver Module - Revision F, 2021. accessed on June 2022.

[17] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *31st IEEE conference on local computer networks*, pages 641–648, 2006.

[18] D. Russo, B. V. Roy, A. Kazerouni, I. Osband, Z. Wen, et al. A tutorial on Thompson sampling. *Foundations and Trends® in Machine Learning*, 2018.

[19] O. Seller and N. Sornin. Low power long range transmitter, Aug. 2014. US Patent App. 14/170,170.

[20] Semtech. SX1276/77/78/79–860 MHz to 1020 MHz low power long range transceiver. Technical Report Datasheet, Mar. 2015.

[21] M. Slabicki, G. Premsankar, and M. D. Francesco. Adaptive configuration of LoRa networks for dense IoT deployments. In *IEEE/IFIP Network Operations and Management Symposium- (NOMS)*, pages 1–9. IEEE, 2018.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[23] D.-T. Ta, K. Khawam, S. Lahoud, C. Adjih, and S. Martin. LoRa-MAB: A flexible simulator for decentralized learning resource allocation in IoT networks. In *12th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 55–62. IEEE, 2019.

[24] B. Teymuri, R. Serati, N. Anagnostopoulos, and M. Rasti. LP-MAB: Improving the Energy Efficiency of LoRaWAN Using a Reinforcement-Learning-Based Adaptive Configuration Algorithm. *Sensors*, 2023.

[25] A. Yousuf, E. Rochester, B. Ousat, and M. Ghaderi. Throughput, coverage and scalability of LoRa LPWAN for internet of things. *26th International Symposium on Quality of Service*, pages 1–10, 2018.