Increased flexibility in long-range IoT deployments with transparent and light-weight 2-hop LoRa approach

Mamour DIOP
University of Pau, France – Gaston Berger University, Senegal
mamour.diop@univ-pau.fr, serigne-mamour.diop@ugb.edu.sn

Congduc PHAM
University of Pau, France
congduc.pham@univ-pau.fr

***Abstract*—LoRa is a recent low-power and long-range sub-GHz radio technology where devices can communicate in 1-hop to a gateway several kilometers away. Long-range radios typically remove the complexity of maintaining a multi-hop network with intermediate nodes for relaying information. Nonetheless, even with the increased range, 1-hop connectivity can be difficult to achieve in real-world deployment scenario, especially for remote and rural areas where density of gateways is low and where devices/gateway are usually deployed for a specific application. This article describes a 2-hop LoRa approach to seamlessly extend a deployed LoRa network in order to reduce both packet losses and transmission cost. We introduce a smart and battery-operated relay-device that can be added after a deployment campaign to transparently provide an extra hop between the remote devices and the gateway. Field tests are conducted to assess network reliability by dynamic insertion of a relay-device between the remote end-devices and the gateway, without advertising its presence. Energy consumption is also discussed.**

***Index Terms*—LoRa, Low-power IoT, Low-cost IoT, Multi-hop, Rural applications**

I. Introduction

LPWAN standing for Low-Power Wide Area Networks is becoming a de-facto standard when deploying IoT infrastructures. Under this broad term, there are a variety of radio technologies for wireless communication over very long distances with battery-operated small devices. The most popular LPWAN technologies (SigFox, LoRa) can achieve more than 20km in line-of-sight (LOS) conditions.

The H2020 WAZIUP project (http://www.waziup.eu) proposes a generic IoT platform using low-cost Arduino board (e.g. Arduino ProMini) and LoRa radios for deploying smarter rural applications in developing countries [1]. From the generic platform, WAZIUP proposes 4 Minimum Viable Product (MVP) on Cattle Rustling, Smart Agriculture, Water-Fish Farming and Waste Management, that have already been deployed in Ghana (Fish Farming, Agriculture-Weather), Togo (Agriculture-Urban), Senegal (Cattle Rustling, Agriculture-Irrigation) and Pakistan (Agriculture-Irrigation). Feedbacks from these pilots highlighted the fact that even with the longer range offered by LoRa, many of these deployment campaigns suffer from connectivity issues with the gateway as clear LOS communication is hardly the case as illustrated by Fig. 1 in maize crop fields. Reasons are numerous. For instance, there are constraints on gateway and gateway's antenna placement: can be limited to the farm office where power supply and wired Internet are available. Some devices can also become very isolated from the vast majority of deployed devices because of field configuration. Regarding the transmission power, there can also be severe limitations in many countries.

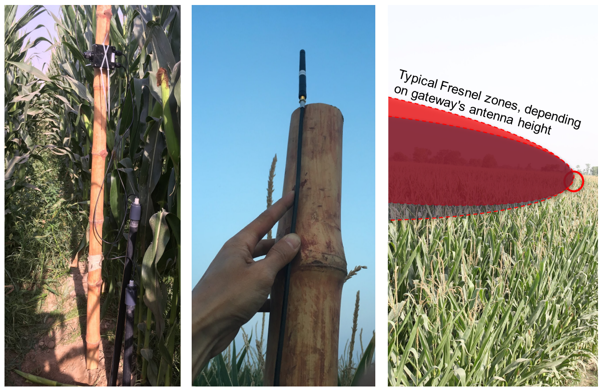
Typical Fresnel zones, depending on gateway's antenna height

Fig. 1: Real-world deployment with NLOS conditions

In this work, a 2-hop LoRa approach is proposed to extend the LoRa network coverage. However, in doing so, the key objective is to design a smart, transparent and battery-operated intermediate node – *relay-device* – that can be added after a deployment campaign to seamlessly provide an extra hop between the remote devices and the gateway. The rest of the article is organized as follows. Section II reviews existing multi-hop LoRa approaches. Section III describes our proposed approach based on low-power relay nodes. Performance evaluation, experimental measurement results are presented in Section IV while discussions are presented in Section V. We conclude in Section VI.

II. Related works

In a multi-hop network, intermediate nodes provide routing and relaying facilities. In exploring the limits of LoRaWAN, the authors in [2] addressed the use of TDMA and multi-hop solutions to reduce transmission power and limit the number of collisions. From there, an extension of LoRaWAN protocol enabling relay-based communication to extend the coverage area without the need of additional gateways is proposed in [3]. There is also LoRaBlink [4] which is a protocol on top of

LoRa's physical layer designed to support reliable and energy efficient multi-hop communications. Time synchronization is used to define slotted channel access and, while downlink messages are distributed through flooding, nodes use a directed flooding approach for uplink communications. In [5], the author analyzed the impact of introducing a forwarder node between an end device and a gateway to improve the range and quality of LoraWAN communications. As the forwarder aims to reduce the power consumption on end-nodes, the work mainly focused on an energy analysis. However, the device receive window must be increase to manage downlink packets. In [6], the authors investigated the combination of LoRa and concurrent transmission (CT) – a recently proposed multi-hop protocol that can significantly improve the network efficiency – to realize a reliable CT-based LoRa multi-hop network. On one hand, the long transmission range of LoRa ensures coverage and reduces the number of redundant relay nodes. On the other hand, the CT protocol helps to realize a simple but efficient one-to-any fast packet broadcast by introducing synchronized packet collisions. [7] proposed a multi-hop uplink solution compatible with LoRaWAN specification, which can act as an extension to already deployed gateways. End nodes transmit data messages to intermediate nodes, which relay them to gateways by choosing routes based on a simplified version of Destination-Sequenced Distance Vector (DSDV) routing.

These works propose centralized approaches controlled by the gateway, the network server or the initiator, which configure both the relay nodes and the device nodes through MAC commands. In addition, the synchronization mechanism requires many message exchanges. In most of these works, end-devices act as relay depending on the needs. [7] introduces routing nodes (RNs) for relaying uplink packets from leaf nodes. However, RN are assumed to be not energy constrained. The purpose of this work is not to use the multi-hop concept to propose a new LPWAN protocol, or an extension of LoRaWAN, to solve the aforementioned problems. In rural applications context for developing countries, gateways cannot act as relays as in [8] where more gateways are deployed to ensure multi-hop communication. This would lead to additional deployment cost since a gateway (a) usually needs an unlimited power source, (b) requires an IP connection to operate and (c), is the most expensive component, even in our low-cost context. End-devices also don't act as relays because they run very specific sensing template code and must be placed according to sensing needs.

III. Smart 2-hop relaying mode

A. Principle

Our 2-hop LoRa relay approach consists in a post deployment addition of an extra hop between some end-devices and the gateway in a non-LOS scenarios. This approach works only for periodic traffic with low data rate applications, and not event-based traffic. We propose to have *relay-devices* which are dedicated low-power nodes with behavior different from the end-devices. However, similar to the end-devices, relay-devices are built from the generic hardware IoT platform but their unique feature is to extend the network coverage by performing data receive and forward operations. It does not take part in any data sensing, data processing nor aggregation tasks. One of the major considerations of a relay-device should be its appropriate location to cover areas where connectivity is either lost or unstable after the network deployment. We designed the relay-device with the following requirements:

- *Low-cost, low power*: a relay-device should use exactly the same hardware used for low-cost end-devices (i.e. Arduino Pro Mini, see Fig. 2) with no additional hardware such as Real-Time Clock which would add complexity and impose different hardware setting. The objective is to make the relay-device only different by software means: an end-device can be "recycled" and reprogrammed to act as a relay-device. As it is also desirable that relay-devices run on battery, their longevity must be similar to the longevity of end-devices. Being battery-operated they must not listen continuously, which basically would make them gateways and this is not what we wanted.

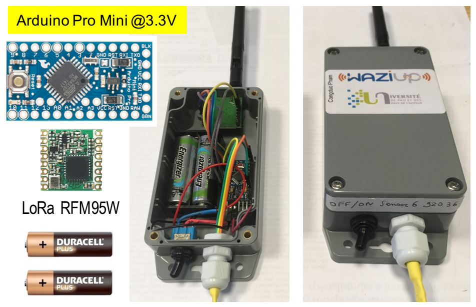

Fig. 2: Low-cost hardware

- *Smart* : relay-devices must be designed to remain in low-power mode most of the time. Obviously, they have to wake-up at appropriate moments to catch uplink transmissions from specific devices in order to perform the relay operation. This is the major consideration of this work since missing uplink packets would make the network less reliable than it was. Therefore, a relay-device must be able to switch from sleep to active mode by smartly analyzing the uplink pattern from end-devices.
- *Transparent*: relay-device nodes must be transparent to the rest of the network: (a) no change in hardware or software for end-devices or gateway to support the new 2-hop approach; (b) no additional signaling traffic between relay-devices and end-devices or gateway. Therefore, end-devices should not be aware of the 2-hop relay mode, nor to perform any discovery and binding process to a nearby relay-device. A relay-device also does not need to exchange parameters with the gateway for advertising its presence. And, on the gateway side, no scheduling mechanism for end-devices and relay-devices is required. The presence of a relay-device should not be detected although it is possible to indicate its presence with a

specific flag in the packet header if it is desirable for the gateway (or network server) to have this information. Our approach is not centralized, neither at gateway nor network server as in related works. Furthermore, withdrawal or failure of a relay-device leaves the network as functional as before its integration in the network.

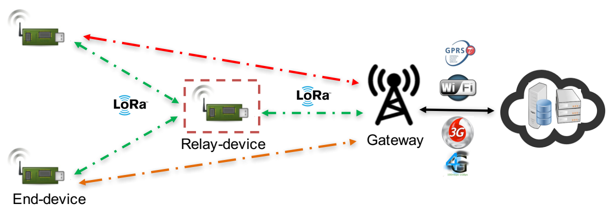

Fig. 3: Long-range 2-hop connectivity architecture

Fig. 3 depicts our proposed architecture for providing a transparent 2-hop LoRa connectivity. The red link means no direct connectivity while the orange link means unstable connectivity. The green links means high quality, stable connectivity. The main advantage of our smart relaying mode is related to the relay-devices' ability to adapt in complete autonomy and transparency to their deployment environment. This is realized with an autonomous and asymmetric synchronization approach. It does not require any time synchronization between the nodes, e.g. end-devices behavior remain unchanged as indicated previously. It is asymmetric in the sense that the synchronization work is done by the relay-device: only the relay-device has to learn wake-up periods of the end-devices.

B. Proposed algorithm

In a typical telemetry LoRa network, end-devices periodically measure environmental parameters and transmit data packets mostly at regular intervals, being most of the time in deep sleep mode where they are unable to send nor receive packets. We assume here that end-devices wake-up at least once every 60 minutes – from their local time as there is no synchronicity between end-devices. When inserted in an existing LoRa network, relay-devices are responsible for forwarding data packets from end-devices with no prior knowledge of how end-devices will wake up. Once deployed, a relay-device discovers end-devices in its vicinity and will build a wake-up table. When powered-on a relay-device first runs an observation phase and then a data forwarding phase.

1) Observation phase: The observation phase consists in observing network traffic for a specified duration D_{obs} to learn the traffic pattern. At start-up, a relay-device usually does not know when it will receive an uplink packet, so it needs to be in receive mode during all the observation duration. This observation duration must be long enough to catch the various uplink packets from end-devices. Assuming that end-devices wake-up at least once every 60 minutes, an observation duration longer than 60 minutes is sufficient. In the observation phase a relay-device receiving an uplink packet from an end-device will (a) records relevant information of the uplink packet such as the source address, the timestamp, etc. and, (b) forwards the packet to the gateway by keeping the original packet header.

Note that packet forwarding from a relay-device to the gateway during this phase can also allow for transmission quality comparison if the original packet also reach the gateway. This process, detailed in Figure 4, is repeated throughout the observation duration. When the observation phase is over the relay-device switches to the data forwarding phase.

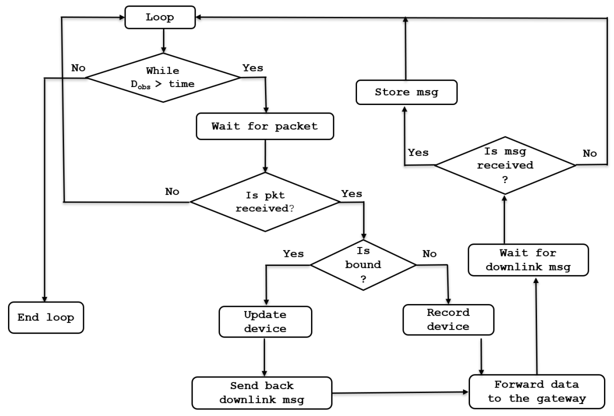

Fig. 4: Observation phase

2) Data forwarding stage: With the collected information during the observation phase, the relay-device is now able to determine wake-up time of the end-devices in its vicinity. It can determine its own activity schedule in each round to wake-up at appropriate moment to catch and forward uplink packets and remain in low-power mode the rest of the time. Algorithm 1 shows how the sleep period is computed.

Algorithm 1 Computing sleep period

```
Input:
    devices: an array of bound devices to relay device, sorted in receiving packet order.
Output:
    sleep_period
1: min_time ← devices[0].timestamp + devices[0].reception_interval − devices[0].toa
2: for $i = 1$ to devices.size() do
3:     tmp_time ← devices[i].timestamp + devices[i].reception_interval − devices[i].toa
4:     min_time ← min(min_time, tmp_time)
5: end for
6: sleep_period ← min_time − current_time
7: return sleep_period
```

In the data forwarding phase the relay-device determines the wake-up time T (using computed $sleep\_period$) to wake up to catch the next uplink packet from device i. Once awake, the relay-device enters in receive mode waiting for the next uplink packet for a bounded amount of time. When receiving the uplink packet it simply forwards the packet to the gateway. Note that upon reception of the uplink packet from device i the relay-device updates the wake-up time of device i accordingly to take into account any clock drift.

3) Handling downlink messages: A relay-device may receive downlink packets from the gateway to specific devices. Usually, an end-device needs to open a receive window after its uplink transmission to wait for an incoming downlink transmission. Here each protocol can define its own timing to make sure that both gateway and end-device are somehow synchronized for the downlink transmission. For instance, in the LoRaWAN specification [9], the end-device opens two receive windows after 1s and 2s after an uplink transmission. In our proposition the relay-device should be transparent and should not add additional delays compared to the original configuration. Therefore, in both the observation and data forwarding phase, when receiving a downlink packet, the relay-device stores this packet and will forward it at the next uplink transmission from the corresponding end-device instead of directly forwarding the downlink packet to the device. The reason is because the end-device receive window does not probably take into account the receive&forward delay introduced by the relay-device.

IV. Experimentations

We performed field tests to assess the performance of the proposed 2-hop approach for increasing network reliability.

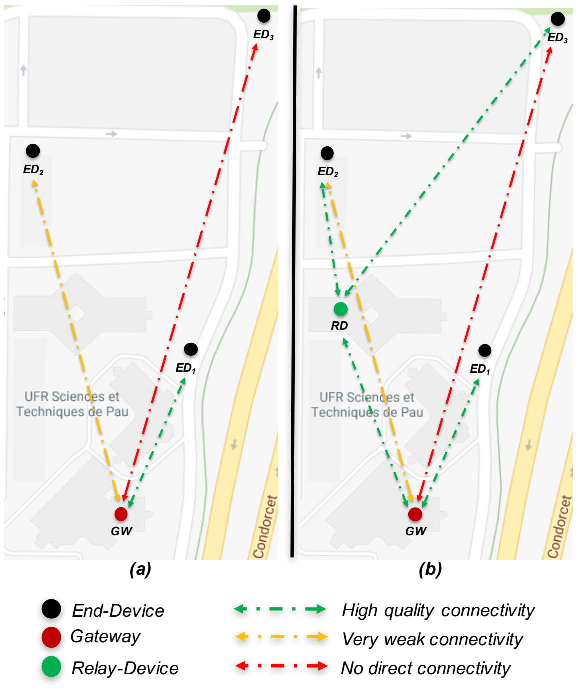

Fig. 5: Deployment scenarios

The university campus with many vegetation and sparse buildings represents our rural environment of deployment. We deployed at first a network consisting of 3 soil humidity end-devices (ED_1, ED_2, ED_3) and one gateway (GW), as shown in Fig. 5(a). GW was placed in the car park of the Faculty of Science and Technology, two meters above the ground. End-devices have different transmission intervals: ED_1 sends a data packet every 3 minutes, ED_2 every 7 minutes, ED_3 every 11 minutes. LoRa parameters of the experiments were chosen as follows: spreading factor of 12, bandwidth of 125kHz and coding rate of 4/5, which is the usual setting that provides the longest range. The transmission power for all tests has been set to 14dBm and all measurements were done in none LOS conditions. Two relevant metrics have been identified: packet error rate and power consumption.

A. Network reliability in 1-hop

In our first set of experiments, we use the classical 1-hop LoRa communication scheme as depicted in Fig. 5(a). In order to determine the network reliability, we simply measured the number of correctly received packets by the gateway GW. Results are as follows: high quality for ED_1 (100% success), weak for ED_2 (40% success), no connectivity for ED_3.

B. Network reliability with 2-hop

In order to assess our 2-hop approach, we introduced a relay-device (RD) in the network so as to obtain stable connectivity between the isolated nodes (ED_2, and ED_3) and RD, but also between RD and the gateway. Fig. 5(b) shows this deployment scenario. We first tested the network reliability by adopting our 2-hop approach with the relay-device in continuously listening mode waiting for uplink packets. As expected, results validated that adding an extra hop between isolated end-devices and gateway can significantly increase the link reliability in high packet error rate conditions. Indeed, all packets sent by end-devices have been correctly received on the gateway side. That means packets from ED_2 and ED_3 were catched by RD and forwarded to the gateway.

C. Investigating end-device/relay synchronization

We also conducted tests to assess the relay-device's ability to automatically synchronize with the rest of the network. Here, the relay-device is not in continuous listening mode anymore and is put in deep sleep mode between 2 expected transmissions from end-devices. The observation phase is set to 25 minutes and at least two packets per end-devices are expected to be catched. We ran each test during 1 hour: the first 25 minutes for the observation phase and the remaining time for the data forwarding phase where the relay-device wakes up from deep sleep mode at various moments T.

We added a guard time, T_{guard}, to investigate the synchronization level between end-devices and the relay-device to catch the next uplink packet. Therefore, instead of waking up exactly at time T, a relay-device will wake up at $T - T_{guard}$. We expect T_{guard} to be small, less that 1s for instance, as the observation phase should provide the relay-device with wake-up schedule informations. We varied T_{guard} from $0s$ to $5s$ we measured the ratio of correctly received packets at the gateway, i.e. uplink packets catched and forwarded by RD to GW.

As shown in Fig. 6, we observed a total desynchronization of the relay-device with the rest of the network when $T_{guard} \leq 2s$. When $T_{guard} \in [3, 4]$, synchronization is partial: at least 50% of packets can be correctly received but not more than 70%. 100% can be reached when $T_{guard} = 5s$.

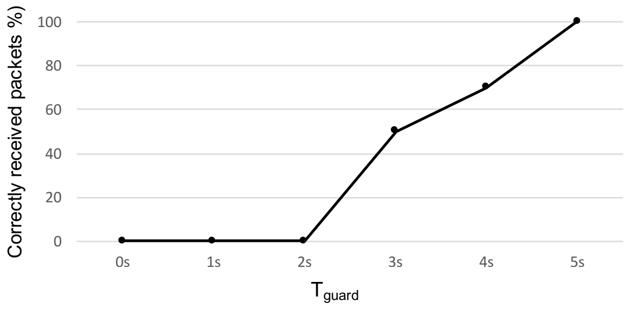

Fig. 6: Percentage of received packets, data forwarding phase

Although a guard time of 5s is tractable, we were quite surprised by these results as we really expected a better level of synchronization even with the native Arduino ProMini millisecond counter. As clock drift is definitely not the reason for the desynchronization, we started investigating this issue in more details.

D. Improving end-device/relay synchronization

First, we have to explain how the deep sleep mode works without the help of an external RTC module. The Arduino's microprocessor (ATMega328P) can be put into a sleep mode with its internal watchdog timer triggering a wake-up after a pre-defined period of time. In doing so, the power savings are impressive in power-down sleep mode (deep sleep) as the board only draws about $5\mu A$. However, as opposed to an RTC which can be programmed to generate a wake-up interrupt for an arbitrarily long period of time, the ATMega328P can only sleep for a maximum of $8s$ (options are 15ms, 30ms, 60ms, 120ms, 250ms, 500ms, 1s, 2s, 4s and 8s) when using the internal watchdog timer. If we need to sleep for a longer period, which is actually the case, then we need to use a loop to sleep for several 8s periods.

The relay-device sleep period between 2 wake-up (to catched uplink transmissions) is therefore divided into a number of sleep cycles of $8s$. We use another Arduino board connected to the relay-device to measure the exact duration of an 8s-sleep cycle that includes the wake-up of the microcontroller and the few instructions needed to operate the loop (the second Arduino is needed as the internal timer is disabled in deep sleep mode). Our measurements showed that the wake-up time after each cycle of $8s$ is increased by $158ms$: instead of a sleep period of $8s$, the relay-device is actually sleeping for $8.158s$. The overhead on an 8s-period is not negligible as accumulation effect is enough to create complete desynchronization: for a sleep period of 6 minutes (i.e. 45 8s-cycles), there is an overhead of $45*0.158s = 7.11s$! Of course the same overhead also applied for end-devices but it is already taken into account in the observation phase. The issue here is when the relay-device itself determines its sleeping period (period between 2 wake-up to catched uplink transmissions) this overhead has not been taken into account.

After identifying this issue, it is easy for the relay-device to sleep for the exact amount of time by taking into account the sleep overhead: the number n of 8s-period is obtained by dividing T by $8.158s$ and not by $8s$. Then, if n is not an integer, the remaining waiting time (which is less than $8.158s$) will be handled by a combination of smaller deep-sleep durations (e.g. 250ms, 500ms, 1s, 2s, 4s and 8s) and also taking into account the additional overheads. After correcting the deep-sleep duration in the relay-device implementation, new tests and measurements indicate a tight synchronization between end-devices and the relay-device where only a safety guard time of $500ms$ is introduced.

V. Discussions

A. Radio duty-cycle

In some countries the transmitter can be constrained by duty-cycle limitations. In Europe, following the ETSI EN300-220-1 recommendations [10], a transmitter is limited to 1% duty-cycle (i.e. 36s/hour) in the general case, even if it can change to another frequency channel. A relay-device forwarding uplink packets from n end-devices will have to transmit at least n packets/hour. Assuming each transmission takes about 1.5s (approximatively the time-on-air of a 20-byte payload packet – header included) then a relay-device can relay 24 packets/hour which is quite sufficient in most cases.

B. Energy consumption

The Arduino Pro Mini (in its 3.3v and 8MHz version) with the LoRa module draws about 40mA when active (taking a measure) and transmitting. The whole process takes about 2s. In deep sleep mode, the board draws 5uA. Therefore an end-device sending 1 measure/hour consumes in the average $(2 * 40mA + 3598 * 0.005mA)/3600 = 0.0272mA$. We have real devices running on AA batteries that have been functioning for more than 2 years at time of writing.

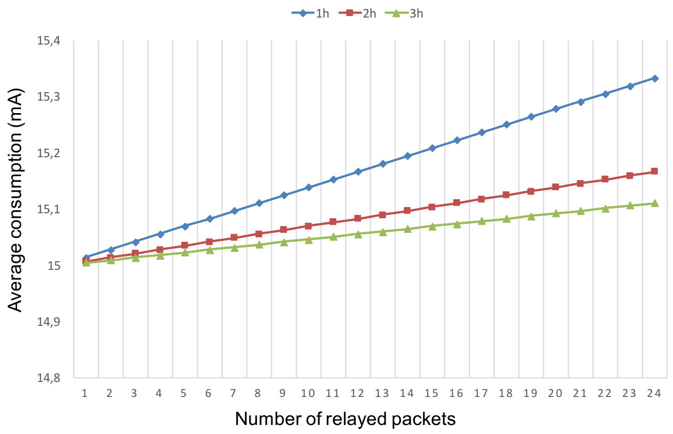

Fig. 7: Average consumption of the observation phase

1) Observation stage consumption: In the observation phase, a relay-device must remain in continuous receive mode for a specified duration D_{obs}. The Arduino ProMini running at 3.3v consumes about 15mA in receive mode. Then, it has to forward the packet. Energy consumption is similar to the transmission from an end-device, i.e. 40mA during 2s. At the relay-device level, managing 3 isolated end-devices by relaying for example 3 packets for a duration D_{obs}=60 minutes, consumes in average $((3*2s)*40mA+(3600s-3*2s)*15mA)/3600s = 15.04mA$. Figure 7 shows the average consumption of a relay-device when relaying n packets during 1 hour, 2 hours and 3 hours. Results shows that 1 hour of observation has little impact on the battery lifetime even when relaying the maximum number of packets per hour allowed by the ETSI radio duty-cycle (n=24).

2) Data forwarding stage consumption: Regarding the relay-device, it has to wake-up and forward uplink packets. For each wake-up, there will be a continuous receive for a maximum of 1s, then it has to forward the packet during 2s. Therefore, for each uplink packet, the relay-device consumes in the average $(1s*15mA+2s*40mA)/3s = 31.66mA$. If we assume that a relay-device is used to relay a very small number of isolated end-devices, e.g. 3 end-devices, then the number of wake-up can be limited. For instance, with 3 end-devices, the relay-device has to wake-up 3 times per hour resulting in an average consumption of $(3*3s*31.66mA + (3600s - 3*3s)*0.005mA)/3600s = 0.084mA$ which still allows for more than 3 years of operation. As illustrated in Fig. 8, results are even better when the relay-device has to wake-up 3 times every 2 hours (more than 6 years of operation) or every 3 hours (more than 9 years of operation). For about 1 year of operation, a relay-device can relay 11 packets if it has to wake-up every hour, 22 packets every 2 hours,…

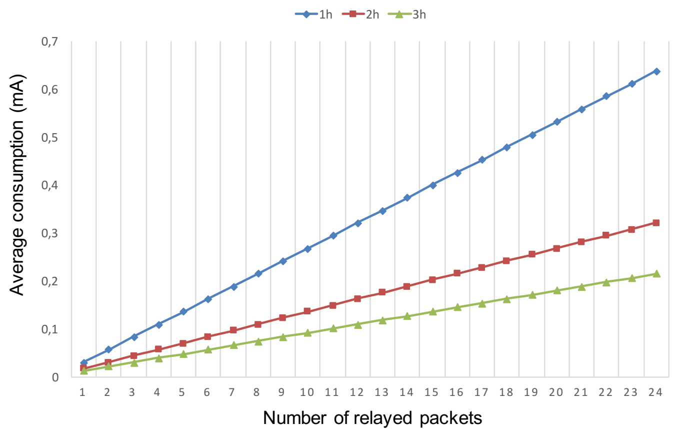

Fig. 8: Average consumption of the data forwarding phase

C. Insertion of new isolated end-devices

The insertion of new isolated end-devices is done by simply resetting the relay-device which will go through a new observation phase. Although this approach will require a new observation phase where the relay's radio will be in continuous receive mode, it appears to be the most transparent approach, avoiding control message exchanges between end-devices, gateway and relay-device. When assuming that the insertion of new isolated devices is also a very rare event, the cost of the observation phase as discussed above has little impact on the relay-device's longevity.

D. Desynchronization due to packet collisions

In the unlikely event of packet collision (because it is assumed that the number of isolated end-devices is small), and if there are some CSMA-like mechanisms implemented, end-devices involved in the collision will have their transmission delayed. Actually the receive window at each wake-up can be set to several seconds as a packet reception will close the window. As indicated previously, upon reception of an uplink packet from device i at time t the relay-device updates the wake-up interval for device i accordingly to take into account any clock drift. In case no packets is received from device i for several consecutive periods, the relay-device will initiate a new observation phase.

VI. Conclusion

We presented a 2-hop LoRa approach to increase reliability in real-world deployment scenarios. We proposed a smart, transparent and low-power relay-device that can be added seamlessly into an existing LoRa network, between some isolated end-devices and the gateway. Both end-devices and gateway are unchanged and can work with or without the relay-device. The experimental tests demonstrate the effectiveness of our approach, especially validating the relay-device's ability to synchronize in an automatic and asymmetric manner with the rest of the network. Using low-cost hardware for the relay-device, the experimental tests also show that a safety wake-up of $500ms$ prior to the expected time of receiving an uplink packet is sufficient to significantly increase the network reliability.

Acknowledgments

This work is supported by the WAZIUP project funded by EU Horizon 2020 research and innovation program under grant agreement No 687607.

References

[1] C. Pham, A. Rahim, and P. Cousin, "Low-cost, long-range open iot for smarter rural african villages," *Proceedings of the IEEE International Smart Cities Conference (ISC2), Trento, Italy*, 2016.
[2] F. Adelantado, X. Vilajosana, P. Tuset, B. Martinez, J. MELIA-SEGUI, and T. Watteyne, "Understanding the Limits of LoRaWAN," *IEEE Communications Magazine*, 2017.
[3] A. Sanfratello, "Enabling relay-based communication in lora networks for the internet of things: design, implementation and experimental evaluation," Master's thesis, University of Pisa, Italy, 2016.
[4] B. Martin, V. John, and R. Utz, "Lora for the internet of things," *In Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN)*, pp. 361–366, 2016.
[5] B. V. D. Velde, "Multi-hop lorawan: including a forwarding node," 2017.
[6] C.-H. Liao, G. Zhu, D. Kuwabara, M. Suzuki, and H. Morikawa, "Multi-hop lora networks enabled by concurrent transmission," *IEEE Access 5*, pp. 21 430–21 446, 2017.
[7] J. Dias and A. Grilo, "Lorawan multi-hop uplink extension," *Computer Science 130*, pp. 424–431, 2018.
[8] D. Lundell, "Ad-hoc network possibilities inside lorawan," Master's thesis, Lund University, Sweden, 2017.
[9] LoRa-Alliance, "Lorawan specification, v1.0.2," 2016.
[10] ETSI, "Electromagnetic compatibility and radio spectrum matters (erm); short range devices (srd); radio equipment to be used in the 25 mhz to 1 000 mhz frequency range with power levels ranging up to 500 mw; part 1: Technical characteristics and test methods," 2012.