ActLoRa: Supporting Actuators in LoRaWAN

Dimitrios Zorbas
Nazarbayev University, School of Engineering & Digital Sciences, Nur-Sultan, Kazakhstan
Author's email: dimzorbas@ieee.org

***Abstract*—This paper examines the problem of supporting actuators in a LoRaWAN network. It proposes a novel autonomous mechanism to allow one-to-one and one-to-many direct communication between different nodes of the same network for actuating purposes. The proposed scheme, called ActLoRa, uses a synchronized subnetwork which coexists with LoRaWAN while the end-devices can still register to the same network following the typical over-the-air activation process consisting of a join-request and a join-accept message. ActLoRa is evaluated using simulations and testbed experiments. The simulation results show that the performance of the synchronized part of the network is not considerably affected even when the Aloha-part consists of many nodes. The experiments are used to show the feasibility of the approach through the development of a proof-of-concept.**

I. Introduction

Low Power Wide Area Network (LPWAN) protocols have been recently proposed to support applications that require long range transmissions combined with low energy cost. LoRaWAN is one of the top LPWAN protocols currently used in many different application domains such as in smart agriculture, in smart cities, and in industrial automation [1]. LoRaWAN relies on the LoRa physical layer, a proprietary Chirp Spread Spectrum (CSS) radio technology. LoRa can achieve transmission ranges of several kilometers with Line-of-Sight (LoS) while it presents remarkable resilience against interference, fading, and Doppler effects.

LoRaWAN creates a star-of-stars topology where end-devices can reach one or more gateways with a single hop, while each gateway is connected with a back-end – usually cloud-based – network consisting of base stations that provide services to the end-devices through the gateways. The end-devices' job is to periodically report data related to their vicinity. Alternatively, the end-devices are triggered by events that may happen in their vicinity. The majority of LoRaWAN nodes communicate with the gateway(s) by transmitting packets using a number of uplink radio channels. Downlink communication is used only for data acknowledgements and commands related to the functionality of the network.

Apart from the lack of downlink communication between the gateways and the end-devices, LoRaWAN lacks of a mechanism that will allow nodes of the network to directly communicate with each other. This type of communication is very desirable in applications where actuation is required. An example of such an application is the following: a node (instigator) detects a fire in a building and transmits a packet to one or more nodes (actuators) that control fire extinguishers, evacuation door switches, and fire alarms. With the current LoRaWAN technology, this scenario is not feasible unless the actuators are connected to a separate non-LoRaWAN network (e.g., Zigbee or a wired network). However, this implies additional costs and eventually extra delays because the instigator's packet needs first to be forwarded through the LoRaWAN network to the cloud server, and then from the cloud server to another network server which controls the actuators, and from there to finally reach the actuator devices.

To solve the aforementioned issue, this paper proposes ActLoRa. In ActLoRa, an instigator can directly communicate with one or more actuators of the same LoRaWAN network without the intervention of a gateway or a separate network. The approach works with the minimum possible pieces of information. The network administrator just needs to set up the node dependencies (i.e., groups of nodes that participate in the actuation) on the network server and, optionally, a key for data encryption on the nodes. The rest of the information such as how the nodes are synchronized, at what time they communicate, and how acknowledgments are sent between nodes of each actuating group, is done through the ActLoRa autonomous mechanisms.

ActLoRa bases its operation on TS-LoRa [2], a time-slotted protocol for industrial IoT applications. The time is divided in repeated frames consisting of slots, while each slot accommodates a unique data transmission or acknowledgment. The nodes periodically wake-up to synchronize with the network through the broadcast of gateway beacons. The synchronized nodes that are used for the actuation may co-exist with typical LoRaWAN Class A nodes (see Section III). Section IV presents details of how this scheme works focusing on ActLoRa autonomous slot generation mechanism. The approach is evaluated using simulations and experiments; assessment results are presented in Section V. Section VI concludes the paper and presents plans for future work.

II. Related Research

Due to the very high number of publications related to LoRa/LoRaWAN, this section is limited to works that are closely related to the proposed solution. For a more exhausted literature review, the reader may refer to recent surveys [3, 4].

Machine-to-Machine (M2M) communications is required by many IoT applications especially those with strict delay constraints. Sensor actuation is a primitive M2M concept introduced in the literature more than a decade ago [5], however none of the existing LPWAN protocols support it. Multi-hop industrial IoT protocols such as the 6TiSCH and the WirelessHART are the only solutions that can currently be used for this purpose, however, in the LoRa context (e.g.,

[6, 7]) they usually suffer from high overhead [8] and they cannot run over LoRaWAN.

On the other hand, time-division and network synchronization are two concepts that have been recently adopted by many approaches in the literature with the aim of increasing the network reliability. Most of them rely on a slotted-Aloha scheme where the nodes randomly select a timeslot to transmit their data [9, 10]. Some other approaches propose collision-free and low-overhead designs to increase the efficiency for a known number of end-devices [2, 11]. This is done by scheduling the transmissions in slots. Approaches that combine time-division with carrier sensing mechanisms are also presented [12]. Finally, time-division without the requirement of time-synchronization is proposed by Finnegan *et al.* [13] to double the number of reporting nodes for a single gateway.

Even though these approaches come up with solutions for building more reliable LoRaWAN networks, they are not designed to support actuating nodes. The main drawback is that no mechanism to allow several nodes to wake-up simultaneously, exchange packets with each other and send acknowledgments is provided.

III. Preliminaries

This section briefly presents the main features and functionalities of LoRa and LoRaWAN that are necessary for non-experts to understand the main concepts of this work.

A. LoRa & LoRaWAN

In LoRa all the available channel bandwidth is used while the chirps are spread diagonally during the transmission. The amount of spread to use is decided by a parameter called Spreading Factor (SF) which typically ranges from 6 to 12 (even though SF6 is not practically used). The higher the SF, the longer the transmission time and, thus, the energy consumption. However, a longer range can be achieved with higher SFs. At the physical layer, a LoRa packet consists of a preamble followed by the payload.

The LoRaWAN protocol [14] sits on top of LoRa at the MAC and link layers and it also provides end-to-end encryption, back-end connectivity, and an adaptive settings management technique, called ADR, which allows the nodes to use the least energy consuming settings. In contrast with the LoRa physical layer, LoRaWAN is open source. Three classes of nodes exist; Class A, Class B, and Class C. Each class has its own features and use cases, but class A is the dominant one. All LoRaWAN devices support bi-directional communication, however, the main differences between classes are the downlink communication time window and the energy consumption. In Class A, uplink communications can be initiated at any time by the end-device, followed by two receive windows. Class B devices open downlink communication periodically and synchronized, while Class C devices open the downlink communication channel continuously, and thus, they require a non-intermittent power supply.

Sub-GHz LoRa-enabled commercial devices are obliged to follow strict radio duty cycle regulations imposed by local spectrum authorities. For example, in the EU, the spectrum is divided in bands and each band has a number of channels that are used for communication. Most of the bands have a total duty cycle of 1% per hour which allows the nodes to transmit up to 36 seconds within an hour. Another band which is usually used for downlink communication has an up to 10% duty cycle.

B. Security in LoRaWAN in a Nutshell

All end-devices need to be activated (register) to become part of a LoRaWAN network. The protocol allows two ways to achieve that; the Activation-By-Personalization (ABP) and the Over-The-Air-Activation (OTAA) methods.

In the first case, a network administrator needs to store an application Session Key (`AppSKey`) and a network session key (`NwkSKey`) in a non-volatile memory of the devices. Both keys are unique per device and are used for encryption and authentication, respectively. Apart from these two keys, a unique device address (`DevAddr`) is assigned to each node which is used to identify each node in the network. `DevAddr` is a concatenation of a 7-bit address prefix (`AddrPrefix` – the same for all the nodes in the network) and a 25-bit arbitrary generated number.

In the second case, the generation of `DevAddr` and of the two aforementioned keys is done "over-the-air". Each node that wants to join a network sends a join-request message consisting of its `DevEUI` (unique device identifier) and a network join identifier (`JoinEUI`) which identifies the network server and it is known to all nodes. Join-request messages are sent unencrypted but are followed by a message integrity check (MIC) signed with a unique (per node) application key (`AppKey`) which is stored in a non-volatile memory along with some other information used to avoid replay attacks. The back-end servers (through a gateway) reply with a join-accept message if the node is allowed to participate in the network (i.e., if `JoinEUI` and MIC are correct). The join-accept message contains the `DevAddr` of the node, the network ID, and some other fields related to the functionality of the network and to the replay attack avoidance. Join-accept messages are encrypted with the node's `AppKey`. Once the node gets the join-accept message, it can generate both `AppSKey` and `NwkSKey` autonomously. The back-end servers do the same process so both ends generate the same keys. The keys are valid for the duration of the session.

IV. Enabling Actuation in LoRa Networks

Two conditions need to be met in order to support actuators in a LoRa network. First, two or more nodes need to agree on the exact time that they shall turn their radio on to transmit or receive the actuating data. Second, all these nodes must retain the same key material for their in-group communications. These two conditions are described in the following subsections.

A. Time Synchronization & Frame Structure

In order to simultaneously turn on their radio to transmit or receive data, the nodes first need to be synchronized with the

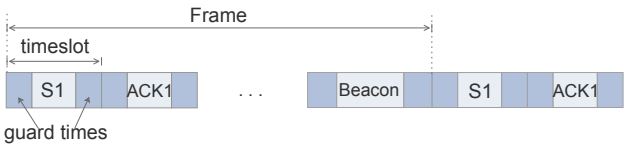

Fig. 1. An example of the ActLoRa frame structure for two synchronized nodes, namely one instigator and one actuator.

network. This can be done either over the LoRaWAN Class-B beacon broadcast mechanism or over another application layer mechanism as the one implemented in [2]. If we assume that the actuation is done over a LoRaWAN network, only the group of nodes who participate in the actuation are required to be synchronized in order to transmit or receive data. The rest of the nodes can follow the default Aloha-based approach of LoRaWAN.

As it is illustrated in Fig. 1, the time between two successive beacons is divided in equally sized slots. Each slot accommodates the transmission of one node, while empty time space is added before and after the expected transmission time to tolerate clock drifts (guard times).

B. Wake-up Time

The process of computing the exact wake up time of the nodes is partially based on the autonomous slot assignment of TS-LoRa, however, the same process in ActLoRa is much more sophisticated. ActLoRa allows all the nodes of an actuating group (an instigator and a number of actuators) to wake up at a specific timeslot to transmit or receive data. A number of slots is reserved after this timeslot which the actuators can use to send acknowledgments. The process does not require any specific knowledge of the group participants, other than their DevEUIs. Algorithm 1 presents the pseudocode of the approach while the following paragraphs describe each step of the approach in detail.

Algorithm 1: Instigator's DevAddr generation.

```
   input: slot, S, K, DevAddr_i ∀ i ∈ K
1  k ← |K|;
2  if k mod 2 = 0 then k ← k + 1 ;
3  DevAddr ← NULL;
4  while DevAddr = NULL do
5      generate a random DevAddr (repeat if already
         generated);
6      if int(hash(DevAddr)) mod S = slot then
7          forall i ∈ K do
8              DevAddr_i ← XOR(DevAddr,
                DevEUI_i[32..63]);
9              ACKsl_i ← slot + 1 + DevAddr_i mod k;
10         if ∃ (i, j) ∈ K, i ≠ j : ACKsl_i = ACKsl_j then
11             DevAddr ← NULL;
12     else
13         DevAddr ← NULL;
14 return DevAddr;
```

As it was mentioned in Section III-B, during the over-the-air node activation, the network server replies to a join request with a join-accept message which among other fields it contains a unique 32-bit address of the joining devices in the network (i.e., DevAddr). The default LoRaWAN computes a large part of DevAddr at random. On the contrary, at the first step, ActLoRa pseudo-randomly generates a DevAddr for each instigator using the following formula:

$$\texttt{DevAddr} = \text{hex}(\texttt{AddrPrefix} \,|\, \text{bin}(k) | \text{rand}(25)), \quad (1)$$

where AddrPrefix is a 4-bit network address prefix, k is the number of actuators in the group (3 bits), and rand($\cdot$) is a function which randomly generates $\cdot$ bit numbers.

The generated DevAddr must first satisfy the following condition:

$$\texttt{slot} = \text{int}[\text{hash}(\texttt{DevAddr})] \ \% \ S, \quad (2)$$

where slot is the desired slot number, hash() is an one-way hash function, and S is a fixed big integer which represents the maximum acceptable frame size in slots. The joining instigator can derive the same slot number by repeating Eq. (2) once it gets the join-accept packet. It must be noted that the instigator's DevAddr must satisfy some additional requirements which are explained in the following subsections.

In ActLoRa, all participants of an actuating group need to wake-up at the same slot. To do so, all k nodes need to receive either the same DevAddr with the instigator or k different DevAddrs all of them pointing to the same slot. The first option is not possible because the device addresses in LoRaWAN are unique. The second option needs a lot of investigation because, even though it is easy to generate multiple DevAddrs which in turn deduce the same slot [2], for reasons explained in Section IV-C the actuators need to be aware of the instigator's DevAddr. An additional reason is that a DevAddr is used in both the header of any data packet and in the Message Integrity Code (MIC) which assures message authentication. Thus, the question is how the network server can generate DevAddrs for the actuators that somehow "include" the DevAddr of the instigator and, at the same time, all of them point to the same slot.

ActLoRa answers the aforementioned question by allowing the network server to blend DevAddrs that are based on the DevAddr of the instigator with the unique identifier (DevEUI) of each actuator. The following formula is used for this purpose for all i in K:

$$\texttt{DevAddr}_i = \text{XOR}(\texttt{DevAddr}, \texttt{DevEUI}_i\texttt{[32..63]}). \quad (3)$$

In fact, the network server calculates an exclusive OR of the instigator's DevAddr and the 32 least significant bits of the receiver's identifier. The least significant bits are used (and not the most significant ones) because the first 24 bits are the same for devices of the same manufacturer (OUI-defined bits). Thus, the probability of having two or more devices with the same 32 most significant bits is quite high. On the contrary, it is impossible to have devices with the same 32 least significant bits in the same network. That practically means that every $\texttt{DevAddr}_i$ of Eq. (3) is also unique. Each actuator i can use $\texttt{DevAddr}_i$ to derive the DevAddr of the instigator and, then, generate the same unique slot using Eq. (2). This can be done by simply executing the reverse operation:

$$\texttt{DevAddr} = \text{XOR}(\texttt{DevAddr}_i, \texttt{DevEUI}_i\texttt{[32..63]}). \quad (4)$$

At this point, all group participants have joined the network and have received a unique DevAddr. All of them can generate the same slot using inexpensive bit operations.

C. Data Transmission

Once the nodes register to the network, they know their slot and they have their own AppSKeys. They also periodically wake up to receive a beacon from a gateway and adjust their clocks accordingly. The data transmission happens at the slot-th slot as it is derived by Eq. (4). The instigators turn on their radio in transmitting mode while actuators turn on their radio in receiving mode during that timeslot. Both sides have to use the same SF and the same radio channel from the list of the channels received during the registration (this is contained in join-accept packet). A frequency hopping mechanism can be applied to avoid jammed or constantly busy channels. Such mechanisms are out of the scope of this paper. Since the actuators cannot be aware if the instigator will transmit or not a packet in every round, they have to wake-up and turn on their radio for some short time until they go back to sleep.

Every actuator that receives a packet from the instigator needs to report its reception in one of the subsequent slots. The number of subsequent slots dedicated for acknowledgements depends on the size of the actuating group. A new question that rises is how the actuators know at what timeslot the acknowledgment will be sent. ActLoRa solves this issue by allowing each actuator to compute its unique acknowledgment slot (ACKsl) as follows:

$$\texttt{ACKsl}_i = \begin{cases} \texttt{slot+1+DevAddr}_i\texttt{\%k if k\%2=1}, \\ \texttt{slot+1+DevAddr}_i\texttt{\%(k+1) if k\%2=0}, \end{cases} \quad (5)$$

where slot is given by Eq. (2) and $k = |K|$. The second part of the addition with the modulo operation must generate a unique number in the range $[0, k-1]$ for all k actuators. The modulo divisor must always be an odd number in order to assure that any combination of dividends and divisors will lead to either an odd or an even slot number (otherwise there is a high risk of infinite loop when two different DevAddrs constantly generate the same slot).

Since $\texttt{DevAddr}_i$ is strongly connected with the instigator's DevAddr (through Eq.(4), the latter must satisfy Eq. (5) for all actuators in K during its generation in Eq. (2). Since $|K|$ is a 3-bit number, the instigator's along with the actuators' DevAddrs can easily be generated in a short time as it is depicted in Fig. 2. In this figure, it is assumed that all nodes in the network are either instigators or actuators. Each instigator has 5 actuators. This scenario is most likely unrealistic because in real applications most of the nodes just report data and do not actuate. However, this scenario is computationally much harder than having a high number of reporting nodes. The longest instigator-actuators tuple generation that was recorded took not more than 0.25 sec on an Intel i7 at 2.66GHz machine.

Each actuating group reserves $|K|+1$ or $|K|+2$ slots in the frame structure. The slots are given to the nodes serially as long as they register to the network. The first slot of each group is given to the instigator. Since k is a 3-bit number, up to 8 actuators per instigator are supported. The model can support more actuators if one more bit is given for k in Eq. (1) and assuming that not more than 2^3 LoRaWAN networks exist in the same area all of them having a different AddrPrefix.

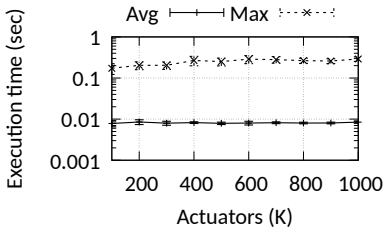

Fig. 2. Average and maximum execution time per instigator-actuators tuple. Each tuple consists of 1+5 DevAddrs. Thus, it is assumed that 20% of the slots are occupied by instigators and 80% by actuators.

It must be noted that if the application does not require acknowledgments or if only one actuator exists in the group, ActLoRa can be simplified by eliminating Eq. (5). In that case, the DevEUIs of the nodes are not required to be known in advance. Moreover, it is not necessary the instigator to register before the actuators because the DevAddrs can be pre-computed for a given (non-serial this time) range of slots.

D. Encryption Key Material

As it is described in Section III-B, each node of the network has in its possession an encryption key (AppSKey) which is used for its one-to-one communication with the gateway (uplink or downlink). Since this key is unique for each node, the nodes in the actuating group cannot communicate with each other. It must be noted that LoRaWAN does not provide any group-key distribution or generation mechanism (not even for beacon-based Class-B devices). It is mentioned that this has to be done either during the device personalization (ABP) or through the application layer.

ActLoRa uses different keys than the default session keys for the in-group communication. Since the nodes do not have enough common information to construct such a key, all the nodes in the group must store a 128-bit randomly generated actuating key in a non-volatile memory similarly to LoRaWAN Class-B nodes. It must be noted that all key and device address generation process can be bypassed if the ABP activation is used. In that case, the actuating keys are stored in the same secure way AppSKeys are stored on each device.

V. Evaluation & Discussion of the results

ActLoRa was evaluated using simulations and testbed experiments. On one hand, the purpose of the simulations was to assess the proposed solution in coexistence of many LoRaWAN nodes and, thus, measure its performance when there is a high probability of collision between packets of the synchronized part and packets of the Aloha-based part of the network. On the other hand, the purpose of the experiments was to test the implementation feasibility of the approach through the development of a proof-of-concept.

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Nodes	100 – 1000
Gateways	2 (Aloha) + 1 (Synchronized)
Terrain size	1000x1000 m
Nodes & Gateways positions	Random
Spreading Factors (Aloha)	7 – 12
Spreading Factor (Sync.)	9
Channel bandwidth (BW)	125 KHz
Preamble Symbols	8
Coding Rate	4/5
Receive window 1&2 SF	7–12 & 9
Uplink/Downlink channels	8 / 8+1 or 3 / 3+1
Payload size	16 Bytes
Path Loss model (see [2])	$\overline{L_{pl}}(d_0) = 107$dBm, $d_0 = 40$m, $\gamma = 2.08$, $\sigma = 3.57$
Frame size	128 sec, S=1001
Guard time	25ms
Receiver Sensitivities	Typical Semtech SX1276
Max Tx power & consumption	14 dBm, 75 mA [2]
Rx consumption	45 mA [2]
Aloha packet rate	1 pkt per 5 min
Synchronized packet rate	1 pkt per 128 sec
Retransmissions to drop	8

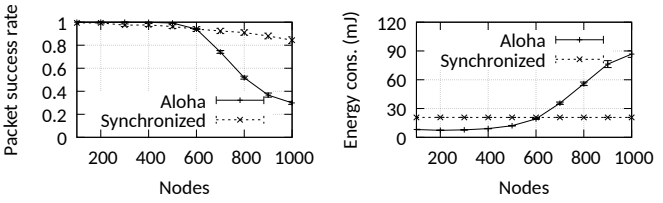

Fig. 3. Packet success rate (left) and energy consumption (right) for a scenario with variable number of nodes, 2 gateways, and 8 uplink radio channels.

A. Simulations

LoRaWAN-SIM[1] [15] was used for the needs of the simulations using variable number of nodes. The network is divided in two parts; the Aloha part and the synchronized part. The majority of the nodes belong to the first category. The nodes of this category send out data with a fixed rate at a random radio channel. Scenarios with 8 and 3 uplink radio channels are examined. An acknowledgment is expected to be received in one of the received windows given the gateways availability. Regarding the synchronized network, the number of instigators is set to 10 and each instigator can have up to 5 actuators. So the maximum number of nodes reserved for actuating is 60. All these nodes communicate through the same (single) radio channel (the first of the 3 or 8 available). The guard times, the clock drift, and the power consumption have been decided experimentally [2]. All the simulation parameters are summarized in Table I. The illustrated results represent averaged values of 50 runs per node instance with different random node and gateway positions. The average SF of the Class-A nodes (after ADR) is just over 9. The imperfect SF orthogonality as well as the capture effect were also taken into account. The 95% confidence intervals are shown in all figures.

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

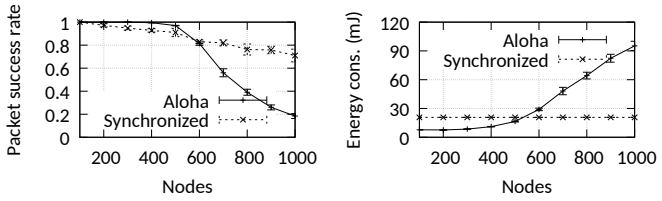

Fig. 4. Packet success rate (left) and energy consumption (right) for a scenario with variable number of nodes, 2 gateways, and 3 uplink radio channels.

The simulation results illustrated in Fig. 3 reveal a low impact of the Aloha-based network on the synchronized network performance. Even when the number of nodes gets high due to retransmissions (because the gateways cannot handle that high traffic volume), the packet success rate of the synchronized network does not fall bellow 82%. On one hand, this can be explained by the higher data rate of instigators which is roughly double than that of the monitoring nodes. On the other hand, two other reasons are the high number of channels and the (on average) higher transmission power of instigators and actuators since no ADR is performed on the synchronized network. Both reasons reduce the probability of collisions. This can also be confirmed by the results of Fig. 4 where only 3 radio channels are used. The reduced number of radio channels negatively impacts the delivery rate of the synchronized network, however, the performance is still acceptable even with very high node populations.

For low node populations, the energy consumption of the synchronized nodes is higher than that of the Aloha network due to the periodic synchronization and the on average higher transmission power. Nevertheless, since most of the energy expenditure comes from the synchronization, the energy consumption is not affected much by the retransmissions. The energy consumption of the Aloha network increases substantially when many nodes are deployed due to the high number of retransmissions.

B. Experiments

ActLoRa was implemented on Pycom Lopy4 devices as an extension of TS-LoRa testbed[2]. One instigator and two actuators were used along with appropriate back-end infrastructure such as a gateway for data reception and beacon transmission, a gateway for join requests, and a Raspberry Pi 4 serving as the network server. The deployment took place in a residential building and the devices were placed indoors. The gateway location was fixed while three different locations were used for the end-nodes with different distances from the gateways; short (RSSI $\geq$–70dBm), medium (RSSI –80 to –90dBm), and long distance (RSSI $\leq$–90dBm) through walls without LoS. The experiments lasted for several hours and were repeated over different days. The parameters of the experiments are presented in Table II.

The experimental results showed that a server response to a join-request message takes on average 27.7ms which mainly accounts for the generation time of the instigator-actuators tuple. Regarding the packet success rate, the results of Fig.

[2]https://github.com/deltazita/ActLoRa

TABLE II
EXPERIMENTAL PARAMETERS

Parameter	Value
Nodes	3
Channel Bandwidth	125 kHz
Preamble Symbols	8
Coding Rate	4/5
Spreading Factor (Data)	9
Frequency	EU868 (1% duty cycle)
Data packet size	16 Bytes
ACK/Beacon packet size	4 Bytes
Guard time	15 ms
Tx power	14 dBm
Packet rate	1 pkt per 18.5sec
Max network size (S)	1001
Data encryption / Hash function	AES-128 / SHA-256

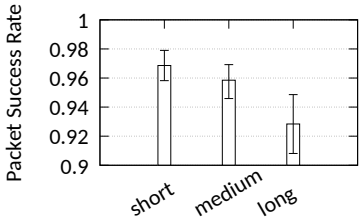

Fig. 5. Experimental results for a scenario with 1 instigator and 2 actuators.

5 reveal a high reliability even when the distance between the instigator and the actuators is long. In fact, most of the retransmissions occurred due to the channel path-loss and almost zero packets were finally dropped.

VI. CONCLUSION & FUTURE WORK

In this work, the feasibility of extending LoRaWAN in order to support M2M communications was examined. The proposed approach, called ActLoRa, implements a synchronized sub-network which can coexist with the other LoRaWAN nodes. ActLoRa is capable of supporting direct communication between one transmitter (instigator) and one or several actuators without prior knowledge of the network but just following a typical over-the-air LoRaWAN activation. Simulation results showed a low impact of the Aloha part of the network on the synchronized one. Experiments confirmed the feasibility of the approach. In the future, ActLoRa will be evaluated experimentally using a higher number of nodes and under different deployment scenarios. A channel-hopping mechanism will also be exploited and be assessed under jamming attacks.

REFERENCES

[1] J. Haxhibeqiri, E. De Poorter, I. Moerman, and J. Hoebeke, "A survey of LoRaWAN for IoT: From technology to application," *Sensors*, vol. 18, no. 11, p. 3995, 2018.

[2] D. Zorbas, K. Abdelfadeel, P. Kotzanikolaou, and D. Pesch, "TS-LoRa: Time-slotted LoRaWAN for the Industrial Internet of Things," *Computer Communications*, vol. 153, pp. 1 – 10, Mar. 2020.

[3] P. Gkotsiopoulos, D. Zorbas, and C. Douligeris, "Performance Determinants in LoRa Networks: A Literature Review," *IEEE Communications Surveys Tutorials*, vol. 23, no. 3, pp. 1721–1758, 2021.

[4] A. Raychowdhury and A. Pramanik, "Survey on LoRa Technology: Solution for Internet of Things," *Advances in Intelligent Systems and Computing*, vol. 1148, no. Ibica, pp. 259–271, 2020.

[5] M. A. Demetriou, "Guidance of mobile actuator-plus-sensor networks for improved control and estimation of distributed parameter systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 7, pp. 1570–1584, 2010.

[6] M. Bezunartea, R. Van Glabbeek, A. Braeken, J. Tiberghien, and K. Steenhaut, "Towards Energy Efficient LoRa Multihop Networks," in *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 1–3, IEEE, 2019.

[7] M. Haubro, C. Orfanidis, G. Oikonomou, and X. Fafoutis, "TSCH-over-LoRA: Long Range and Reliable IPv6 Multi-hop Networks for the Internet of Things," *Internet Technology Letters*, 2020.

[8] D. Zorbas and X. Fafoutis, "Time-slotted lora networks: Design considerations, implementations, and perspectives," *IEEE Internet of Things Magazine*, vol. 4, no. 1, pp. 84–89, 2021.

[9] T. Polonelli, D. Brunelli, A. Marzocchi, and L. Benini, "Slotted ALOHA on LoRaWAN-Design, Analysis, and Deployment," *Sensors*, vol. 19, no. 4, p. 838, 2019.

[10] L. Chasserat, N. Accettura, and P. Berthou, "Short: Achieving Energy Efficiency in dense LoRaWANs through TDMA," in *IEEE International Symposium On a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2020.

[11] 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*, vol. 7, pp. 669–683, Jan. 2020.

[12] A. Triantafyllou, P. Sarigiannidis, T. Lagkas, I. D. Moscholios, and A. Sarigiannidis, "Leveraging fairness in LoRaWAN: A novel scheduling scheme for collision avoidance," *Computer Networks*, vol. 186, p. 107735, 2021.

[13] J. Finnegan, R. Farrell, and S. Brown, "Lightweight Timeslot Scheduling Through Periodicity Detection for Increased Scalability of LoRaWAN," in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pp. 8–15, 2020.

[14] LoRa Alliance Technical Committee, "LoRaWAN™ 1.0.4 Specification." https://lora-alliance.org/resource_hub/lorawan-104-specification-package, Oct 2020. Online; accessed 10-Nov-2021.

[15] D. Zorbas, C. Caillouet, K. Abdelfadeel Hassan, and D. Pesch, "Optimal Data Collection Time in LoRa Networks—A Time-Slotted Approach," *Sensors*, vol. 21, no. 4, 2021.