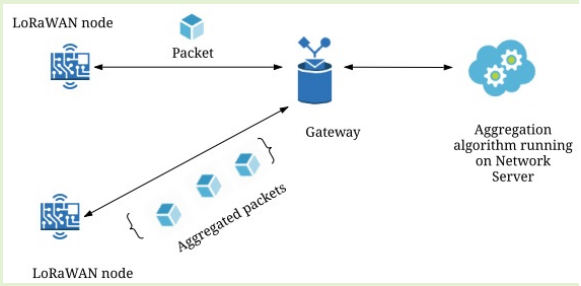
Improving the sustainability of confirmed traffic in LoRaWANs through an adaptive congestion scheme

Jaco Morné Marais, *Student Member, IEEE,* Adnan M. Abu-Mahfouz, *Senior Member, IEEE,* and Gerhard P. Hancke, *Life Fellow, IEEE*

***Abstract*— The scalability of Long Range Wide Area Networks (LoRaWANs) is known to be very sensitive to the presence of traffic from gateways to end devices (downlink traffic). The protocol supports confirmed traffic, for which downlink traffic is generated in the form of Acknowledgements (ACKs). Research has shown that even a limited number of ACKs quickly cause network congestion, negatively impacting scalability. This paper introduces a mechanism, the Adaptive Congestion Scheme (ACS), which aims to monitor the congestion caused by downlink traffic and take steps to reduce it. Currently, the ACS supports one counteraction in the form of a newly developed algorithm called groupedPackets (also introduced in this paper). This algorithm reduces the number of sent confirmed packets by requesting that confirmed nodes (nodes that only send confirmed traffic) aggregate their application packets. Simulations showed that this algorithm improved the successful delivery of both unconfirmed traffic and confirmed traffic. When traffic volumes are low, the algorithm has a minimal impact, but at high network packet arrival rates (higher than 0.1 pkt/s), the successful delivery of especially confirmed traffic increased significantly.**



***Index Terms*— LoRaWAN, LPWAN, Scalability, ACK, ACS, groupedPackets.**

I. Introduction

The recent rise in Internet of Things (IoT) deployments continues to gain momentum, as more and more industries hope to manage their complex supply chains and the high cost of logistics through the use of technology [1], [2]. These deployments require effective Machine-to-Machine (M2M) communication and have resulted in the development of Low Power Wide Area Networks (LPWANs). One such technology, namely LoRa Wireless Area Network (LoRaWAN), was found to be effective for a range of IoT use cases [3], [4].

Particularly, these networks are well suited for smart city deployments due to their support for large amounts of devices (scalability). A LoRaWAN's scalability is influenced by the presence of Downlink (DL) traffic [5], [6]. It has been shown that, amongst other factors, Duty Cycle (DC) restrictions and a high maximum number of transmissions (NbTrans) setting cause a dramatic reduction in network scalability. Their impact is minimal in small networks but occurs in large networks even with a low ratio of confirmed to unconfirmed traffic.

This work aims to reduce congestion in large LoRaWANs through the introduction of the Adaptive Congestion Scheme (ACS). This scheme was developed by the authors and monitors LoRaWANs for the presence of confirmed traffic and congestion before proceeding to activate countermeasure(s). Currently, one countermeasure is available, namely a newly developed algorithm introduced in this paper that aims to improve network performance. This algorithm, referred to as groupedPackets, seeks to reduce the amount of confirmed traffic in a network by requesting that nodes sending confirmed traffic aggregate their application packets. The design and implementation of this algorithm are described in this paper.

Aggregation based techniques have been applied in wireless networks for some time now due to the many challenges these types of networks face [7]. The use of aggregation in LoRaWANs has been mainly explored in the form of Acknowledgement (ACK) aggregation [8]–[11]. Some researchers have also focused on aggregating application data instead, especially in use cases where retrieving sensor data is complicated [12].

This paper focuses on data aggregation in LoRaWANs in a novel way and contributes the following:

Manuscript received 23 September 2021. This research was supported by the Council for Scientific and Industrial Research, Pretoria, South Africa, through the Smart Networks collaboration initiative and IoT-Factory Program (Funded by the Department of Science and Innovation (DSI), South Africa). This work is based on the research supported in part by our industry partner Telkom. The grant holder acknowledges that opinions, findings and conclusions or recommendations expressed in any publication generated by this research are that of the author(s), and that our industry partners accept no liability in this regard.

All authors are with with the Department of Electrical, Electronic and Computer Engineering, University of Pretoria, Pretoria 0028, South Africa e-mail: (jaco.marais@tuks.co.za; g.hancke@ieee.org)

A.M. Abu-Mahfouz is also with the Council for Scientific and Industrial Research, Pretoria 0084, South Africa.

G.P. Hancke is also with the Nanjing University of Posts and Telecommunications, Nanjing 210023, China.

- Discusses how congestion can be monitored in LoRaWANs and introduces the newly developed ACS.
- Introduces and explains how the novel groupedPackets algorithm operates. Unlike static congestion reduction mechanisms, this algorithm will continue to make adjustments when possible.
- Demonstrates through simulations how this algorithm improves network performance.

The paper is organised as follows. Some background on LoRaWAN is provided in Section II. Related work is discussed in Section III. Information on simulation setup is given in Section IV. The impact of congestion is discussed in Section V. Section VI details the methodology behind the ACS and the groupedPackets algorithm. The impact of this algorithm on network scalability is presented and discussed in Section VII. Examples of where this mechanism can be used are given in Section VIII. Finally, a conclusion is given in Section IX.

II. Background: LoRaWAN

The LoRaWAN technology consists of two parts: the Long Range (LoRa) physical layer and the Medium Access Control (MAC) layer protocol called LoRaWAN. LoRa is based on Chirp Spread Spectrum (CSS) and provides different data rates, which represent a trade-off between range and possible bitrate. This is done through the Spreading Factor (SF) parameter, whose options are $\{7, ..., 12\}$ with higher numbers allowing for greater range. A selling point of LoRa is that the SFs are considered orthogonal to each other, allowing transmissions overlapping in time and frequency to be still received successfully. There are, however, some conditions that have to be met [13], [14]. Gateways equipped with LoRa chipsets have eight parallel reception paths to utilise the *quasi* orthogonality of the different SFs.

The LoRaWAN standard [15] dictates the MAC and other protocols for devices utilising LoRa. A star-of-stars topology is used which consists of end devices, Gateways (GWs) and a Network Server (NS), which is a central entity responsible for controlling the network. Maximum application payload sizes are given in [16] and depend on the region of operation. These maximums depend on the data rate used and are derived from the limitations of the PHY layer.

The standard also has an Adaptive Data Rate (ADR) mechanism that allows the NS to adapt and optimise the data rate, channels used and transmit power of static end-devices [15]. This mechanism's main focus is to minimise energy consumption whilst maximising throughput, and is thus not concerned with scalability [17].

III. Related work

The research community has been studying congestion from as early as 2017 ([5]), with numerous new mechanisms proposed to improve scalability [18], [19]. In [18], a synchronisation and scheduling mechanism is proposed that schedules Uplink (UL) and DL transmissions with minimal overhead. The use of groupcasting and geocasting are proposed in [19] to reduce the traffic volumes in a network by specifying which types of sensors must transmit if a specified condition is met in their periodic measurements. These techniques work well in private LoRaWANs, where the sensor data and application are known to the network operator.

The use of aggregation in LoRaWANs has been explored in the form of aggregating acknowledgements [8]–[11]. In [9], an "AggACK" is periodically sent by the NS to ACK the receiving of several device's packets. A similar approach is followed in [10], where a scheduling algorithm dictates when a group ACK will be sent for all packets received in the previous timeslot. The use of timeslots is also explored in [8], [11], where it is combined with aggregated ACKs to improve scalability.

A downside of aggregating ACKs is that if the ACK is not transmitted (DC restrictions) or not successfully received by multiple nodes, all affected nodes will retransmit their packets. In the case of groupedPackets, only one device will be involved, and thus only one transmission will be repeated. However, more ACKs in total will be sent in a system using groupedPackets than in a system aggregating ACKs.

Other approaches to reduce the congestion caused by ACKs are to better manage the use of channels [20], time-slotting [21] or virtually split network resources through slicing [22]. A node focused approach is suggested in [23], in which nodes count the number of retransmissions required by their previous confirmed uplinks and adjust their own DC accordingly. This action is part of the ADC-MAC protocol, which requires nodes to also make similar adjustments based off their payload load and energy levels. Using these techniques, the contention of resources and overlapping of transmission problems are alleviated at the cost of increasing overhead and reducing flexibility.

IV. Simulation setup and scenario

This work extended a popular open-source ns-3 LoRaWAN module called "lorawan"[1]. Its inner workings are well documented in [14], [24] and has since been extended by the team multiple times [25], [26].

The simulation scenario used is a single GW serving multiple nodes, which periodically generate packets with an equal period but random phases. This traffic characteristic matches that of devices in The Things Network (a crowd-sourced LoRaWAN) [27]. The network is configured to follow the default channels and DC limitations of Europe. SF assignment was performed with the SetSpreadingFactorsUp function of the module, which calculates the lowest SF a device should use to still ensure connectivity (ADR is disabled). Results were gathered by averaging over 20 simulations using the sem Python library [28].

A set of 1200 nodes were uniformly distributed in a circular space around the GW with radius $r = 6300\ m$. This value is near the maximum distance a node using SF12 can still communicate when only propagation loss is considered[2]. Nodes generate application packets with a starting size of 10 bytes

[1]Available at `https://github.com/signetlabdei/lorawan`. Work was conducted with developer branch version 159cc5e (4 May 2021).

[2]Older publications using this simulator used 7500 m, see `https://github.com/signetlabdei/lorawan/issues/101` on why this is no longer valid.

to which a random additional i, $i \in \{2, ..., 8\}$ bytes are added. The payload analysis conducted in [27] also found that 50 % of payloads are less than 19 bytes, with the average payload size being 18 bytes. A random element was added as not all devices in such an extensive network can be assumed to be transmitting the same data in every period.

To obtain confirmed traffic ratios, a percentage of devices are selected to send only confirmed traffic, whilst the rest send unconfirmed traffic only. For unconfirmed traffic, only one transmission attempt is allowed per UL packet, whilst for confirmed traffic, the maximum was configured to be eight [3].

A similar approach to [26] was taken to define packet outcomes and performance metrics. For unconfirmed traffic, a packet is considered successful if it is received by a GW who forwarded it to the NS. The case of confirmed traffic is more complicated and can be split into two cases. In the first case, transmission is only considered successful when both the UL and the corresponding DL (ACK) was successfully received. A second, more relaxed case, is also possible where success only requires that at least one of the generated UL packets was successfully delivered to the NS.

Based on the breakdown above, the two performance metrics of interest are:

- Confirmed Packet Success Ratio (CPSR): the probability that a confirmed UL packet was received by a GW and the corresponding ACK was received by the node.
- Uplink Packet Delivery Ratio (ULPDR): the probability that a UL unconfirmed packet was correctly received.

Due to this work's focus on confirmed traffic, the second case for confirmed traffic is not considered. As a result, this work's ULPDR is different from the ULPDR values measured in [26], which considered both cases.

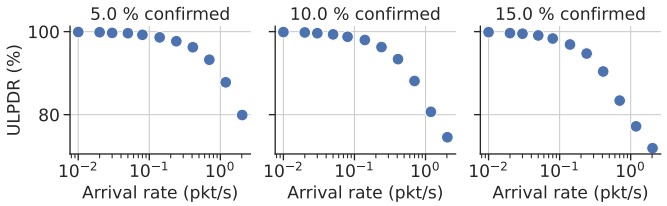
V. Congestion in LoRaWANs

IoT devices infrequently transmit small amounts of data, with data collection aimed at having many devices rather than receiving a specific device's data very reliably. This represents a generic IoT use case such as smart irrigation, however, a selling point of LoRaWAN is its flexibility in supporting different use cases [3]. This is done by supporting downlink traffic in the form of ACKs and downlink data messages to the end device's application. This causes the discussion around scalability to shift from the limitation of GWs to only receive eight simultaneous signals to one in which the impact of downlink traffic must also be considered [29].

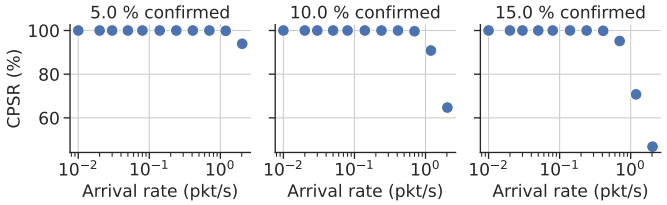
The impact of ACKs on network scalability was explored in [6], which found that the ratio between confirmed traffic and unconfirmed traffic is a key influence on performance. For low ratios (≤ 5 %), the impact can be minimal [30], but for higher ratios, the DC restrictions imposed on gateways cause the performance to drop significantly [27], [31]. The half-duplex nature of LoRaWAN gateways also increases missed UL transmissions as a gateway cannot receive any transmissions whilst transmitting an ACK [31].

In Fig. 1a and Fig. 1b, the performance of unconfirmed and confirmed traffic is shown respectively, demonstrating the impact even low percentages of confirmed traffic have. The graphs show that packet delivery remains above 90 % for network traffic volumes generated by packet arrival rates (λ) of less than 10^{-1} packets per second. After this threshold, a steep decline is experienced for both traffic types, with unconfirmed traffic reducing earlier.

[3]This is the default behaviour of the "lorawan" simulator.



(a) Unconfirmed traffic.



(b) Confirmed traffic.

Fig. 1. Performance of network, examined over several arrival rates and confirmed ratios.

The LoRaWAN protocol will need some improvements to combat the negative impact of confirmed traffic. As discussed in [6], the LoRaWAN protocol does not have a congestion monitoring and response mechanism. This work aims to solve that and has resulted in the creation of the ACS.

VI. Mechanism to Improve Congestion

This section introduces a novel mechanism to improve congestion and is split into three parts. In Section VI-A a method to monitor congestion is introduced with an algorithm aimed at reducing detected congestion given in Section VI-B. Finally, the implementation costs and computational complexity of this mechanism is discussed in Section VI-C.

A. Adaptive Congestion Scheme (ACS)

The ACS monitors a LoRaWAN for congestion before taking action to improve network performance. The scheme is loosely based on LoRaWAN's ADR, and as is the case with ADR, participation is voluntary (nodes can opt-out) with the scheme sending instructions to nodes using a new MAC command (LinkACSReq). Similar to ADR, an answer in the form of a LinkACSAns is sent by the device to indicate if the settings were successfully changed.

In networks with low traffic volumes, congestion control via the ACS is not needed. The ACS can remain dormant until it detects traffic volumes greater than a set threshold in combination with the presence of significant confirmed traffic. For this paper, an examination of Fig. 1a and Fig 1b revealed that 10^{-1} packets per second and the presence of more than 5 % confirmed traffic are good thresholds. In general, these values should be chosen by the network

operator based on desired ULPDR and CPSR levels. For multiple gateway networks, monitoring should be performed at the network level, but the load balancing between gateways should also be considered. The activation of the ACS might be unnecessary if more optimum gateway placements can allow for even utilisation of SFs or better overlap of coverage zones.

To implement the scheme, the LoRaWAN protocol had to be modified. The MAC payload of a message contains the Frame Header (FHDR) with a one-byte long FCtrl element. To add ACS support, this element was expanded to two bytes as shown in Fig. 2. For downlink packets, one bit is used to indicate that the network server can send ACS commands. For uplink packets, one bit is used to indicate a device's participation in ACS. Another three bits are used in uplink packets by the groupedPackets algorithm to indicate how many application packets were aggregated.

The remaining bits are classified as Reserved for Future Use (RFU). At this stage, the novel ACS has only one counteraction: the newly developed request to batch application packets referred to as groupedPackets. More counteractions can be added later and be activated on their own or in conjunction with groupedPackets and thus space was reserved in the header for these.

Bit#	[15..9]	8	7	6	5	4	[3..0]
	RFU	ACS	ADR	RFU	ACK	FPending	FOptsLen

Downlink FCtrl fields

Bit#	[15..12]	[11...9]	8	7	6	5	4	[3..0]
	RFU	GroupedP	ACS	ADR	ADRACKReq	ACK	ClassB	FOptsLen

Uplink FCtrl fields

Fig. 2. New contents for FCtrl of the frame header.

To compensate for a larger header, the maximum application payload size was reduced by one byte to keep the maximum Time on Air (ToA) for LoRaWAN packets unchanged. The maximums are SF dependent, and this change would reduce the maximum allowed payload for packets sent using, e.g. SF12, from 51 to 50 bytes. The number of IoT applications this would impact would be minimal, as analyses performed in [27] showed that 93.7 % of TTN's payloads were below 50 bytes. The larger header size will cause transmissions to take slightly longer and slightly increase power consumption.

Implementation wise, the ACS can be added as another element to the NS similar to ADR algorithms. The meta-data of all packets received by the NS can be analysed to determine the traffic volumes in the network and the ratio of confirmed traffic to unconfirmed traffic.

B. groupedPackets algorithm

This newly developed algorithm seeks to reduce the traffic volumes generated by confirmed nodes by requesting that they aggregate their application packets. This causes nodes to send larger but less frequent confirmed packets. acDC restrictions limit the sending of ACKs by a GW, and by reducing the number of confirmed packets, more ACKs can be sent. This, in turn, reduces the number of retransmissions of confirmed packets, causing fewer packet collisions due to interference. Finally, this also combats the half-duplex nature of GWs, as the number of packets lost due to GW transmissions is reduced as fewer transmissions are required.

Algorithm 1 Pseudocode for NS side groupedPacket.

INPUT: $curNum$: Device's aggregation value;
$device$: a Pointer to a specific end device;
OUTPUT: $newNum$: new recommended aggregation value;

1: $AppPSize \leftarrow 0$ ▷ Max application packet size
2: $TotalAppSize \leftarrow 0$ ▷ Max total application packet size
3: $historyRange \leftarrow 3$ ▷ Num of packets to consider
4: maxGrouping $\leftarrow 5$ ▷ Maximum aggregation limit
5: Update $AppPSize$ and $TotalAppSize$ by calling GetSizes($device, historyRange$)
6: **if** $TotalAppSize + AppPSize \leq 50$ **then**
7: **if** $(curNum + 1) < maxGrouping$ **then** ▷ Aggregation limit not yet reached
8: $newNum \leftarrow curNum + 1$
9: **else**
10: $newNum \leftarrow curNum$ ▷ Limit hit, no changes
11: **end if**
12: **else** ▷ Cannot add another
13: **if** $TotalAppSize > 50$ **then** ▷ Max packet size limit exceeds max, decreasing aggregation.
14: **if** $(curNum - 1) > 0$ **then**
15: $newNum \leftarrow curNum - 1$ ▷ Decreasing aggregation
16: **else**
17: $newNum \leftarrow curNum$ ▷ No changes
18: **end if**
19: **else**
20: $newNum \leftarrow curNum$ ▷ Limit hit, no changes
21: **end if**
22: **end if**
23: return $newNum$

The algorithm contains two parts; one is executed on the NS side to calculate an ideal value for application payload aggregation for each device. The second is executed by each node and ensures that if the aggregation target is too aggressive, as many payloads as possible are sent instead.

All nodes start with no aggregation (only sending the current application payload) whilst the scheme builds up a history of sent packets and their sizes. Once sufficient history has been gathered, currently three packets, the NS side algorithm is enabled and starts making adjustments. The pseudocode for the NS side algorithm is provided in Algorithm 1.

Adjustments to payload aggregation are in increments of one payload to prevent large changes between adjustments (Algorithm 1, lines 8 and 15). The current setting used by a device is extracted from its latest packet's FCtrl element and is used as input $curNum$. There are two limiting factors on how many payloads can be aggregated: the maximum allowed payload size as dictated by the chosen SF and the time delay caused by aggregation may become unacceptable.

The impact of the first factor can be limited through the algorithm estimating the total packet size and, should the

algorithm get it wrong, sending as many packets as possible (node side algorithm). This task is made more difficult because application payload sizes may not be static but vary. Additionally, aggregation requires the addition of a one-byte long delimiter between payloads to allow the application server to separate them.

Total size estimation starts by determining the largest average application payload size in a node's sent history (Algorithm 1, line 5). A test is then done to see if the addition of this largest size and the current payload size($TotalAppSize$) would exceed the maximum allowed (Algorithm 1, line 6). The $AppPSize$ value returned by GetSizes() finds the largest average size by inspecting each packet in the history and calculating its average application payload size. One byte is then added to this total to compensate for the additional delimiter required should the number of packets to be aggregated be increased.

If the combined total remains below the maximum, the node is sent a request to increase the number of payloads it currently groups. Currently, this test uses 50 bytes as the maximum. LoRaWAN does support larger packet sizes, but the maximum depends on the SF used to send a packet. To keep things simple, 50 bytes was chosen to ensure compliance with all SFs. Payloads usually are smaller than this, and this limit would still allow several to be aggregated.

Algorithm 2 Pseudocode for node side groupedPacket.

INPUT: $totalPSize$: Total application packet size due to aggregation + latest payload;
$numGrouped$: The number of payloads aggregated;
$latestPSize$: The size of the latest payload;
$latestP$: The latest payload;
$storedData$: Older payloads stored with delimiters;
OUTPUT: $sentData$: The data to send;

1: **if** $totalPSize \leq 50$ **then**
2: Call SetSentGroupedPackets($numGrouped + 1$) ▷ Update MAC header
3: $sentData \leftarrow (storedData + latestP)$ ▷ Send packet
4: **else** ▷ max size exceeded
5: $newnumPackets \leftarrow 0$ ▷ New number of payloads.
6: $newTotalSize \leftarrow latestPSize$
7: $newData \leftarrow latestP$ ▷ newnumPackets is 0 as 0 here refers to 1 packet being added so far (the latest one).
8: **for all** p in $storedData$ **do**
9: **if** $newTotalSize + size(p) < 50$ **then**
10: $newnumPackets \leftarrow newnumPackets + 1$
11: $newTotalSize = newTotalSize + size(p)$
12: $newData = newData + p$
13: **end if**
14: **end for**
15: Call SetSentGroupedPackets($newnumPackets$)
16: $sentData \leftarrow newData$ ▷ Send packet
17: **end if**

As application payloads vary, a node may not be able to successfully aggregate a newly received payload with the aggregated payloads without exceeding the maximum size. In these cases, the node side algorithm given in Algorithm 2 is activated. To minimise losing payloads, the node side algorithm dictates that a node re-aggregates as many payloads as possible by accessing stored packets in a Last In First Out (LIFO) fashion. This ensures the latest packet is always sent, with the oldest packet(s) possibly being discarded. For example, should three aggregated payloads be 32 bytes with a newly created payload of 20 bytes, a node will be unable to transmit all of them. For this example, assume that each of the aggregated payloads was 10 bytes each, with the delimiters each requiring one byte. The node will send the latest payload (20 bytes) as well as two of the previous payloads for a total of 20 + 1 + 10 + 1 + 10 = 42 bytes. Only the oldest payload was lost in this case, as the maximum of 50 bytes must be adhered to. Lines 6 to 8 of Algorithm 2 show how the latest packet is selected first before the stored packets are examined and added in lines 10 to 13. A payload and its delimiter are stored together in *storedData*.

The aggregation can be too aggressive if payloads are less than, e.g. 5 bytes, as it will take a lot of aggregation before the maximum size is reached. To prevent long delays, the maximum number of packets to be aggregated can be set by the application developer/network administrator, which is five in this case (Algorithm 1, line 4).

C. Implementation costs and computational complexity

In terms of computational complexity, Algorithm 1 will be executed N times, where N is the number of successfully received confirmed packets from devices using groupedPackets in a time period of interest. Line 5 is a call of the GetSizes function, which contains a for loop traversing through $historyRange$ amount of stored packets. This value is fixed, not dependent on N, and cannot exceed 25 (25 payloads of 1B + 24 delimiters = 49 B). All algorithm lines will execute in O(1), as they do not depend on N. For the NS, the algorithm thus is considered linear, O(N), as it will be executed N times.

When considering the complexity of Algorithm 2, the for loop must be considered. This loop will execute in the worst case X times, where X is the length of storedData's array. This length is limited to five by default, with the theoretical maximum being 25. Except for line 8, all other statements do not involve the length of storedData and can thus be considered O(1). The total computational complexity of Algorithm 2 is thus also linear, O(X), with X being limited to no more than 25.

The addition of these algorithms will have an implementation cost. Adding ACS and groupedPacket support is minimal as it was based on the ADR scheme, and the NS already stores information on received packets for ADR purposes. This same history can be used for groupedPackets by also storing the application payload's length and the groupedPackets bits in the Uplink FCtrl header. Similarly, the NS already interfaces with its ADR component to determine if any LinkADRReq commands must be sent to the device. In addition, the NS needs to interface with the ACS component to determine if any LinkACSReq commands should be added.

A LoRaWAN's NS is typically hosted in the cloud (for which the additional load would be minimal) or on a gateway in the case of smaller networks. An example of a gateway which can easily host a NS which supports groupedPackets is the Multitech Conduit, which has an ARM9 processor, 128X16M DDR RAM and 256 MB of flash memory [32].

On the end-device side, an example is common LoRaWAN chips such as the Multitech mDot or the STMicroelectronics STM32WLE5JC which have at least 256 kBytes of Flash memory and 64 kBytes of SRAM [33], [34]. The implementation of the algorithm requires a minimal amount of additional program memory, and aggregation of packets has a maximum aggregation target size of 50 bytes. This is minimal compared to the kBytes of storage available in both cases.

VII. Results

In this section the simulation results is given comparing standard LoRaWANs with networks utilising groupedPackets. Section VII-A showcases how groupedPackets impact performance over time and Section VII-B indicates how arrival rates impacts the algorithm. How various base packet sizes influences the algorithm is given in Section VII-C and a discussion on the delay introduced by the algorithm is given in Section VII-D.

A. Activation of groupedPackets

The groupedPackets algorithm takes several simulation time periods before settling on an optimum aggregation number. The algorithm must first gather packet history, which will allow it to determine if aggregation is possible. If this is the case, the algorithm has a ramp-up period as adjustments are made in increments of one. A device could thus be able to aggregate four packets, but the algorithm will first request it to group two, then three and finally four. It continues to monitor a device's packet history and may request adjustments (increasing or decreasing aggregation) depending on the history.

The algorithm initially requires a three packet history, but this can be reached quickly as packets' retransmissions are included in the history. As a result, the algorithm can send changes to some devices before simulation period three is achieved.

The algorithm's behaviour can be seen in Fig. 3, which shows how the performance for both types of traffic improves over time. The x-axis is the simulation's running time, expressed in periods. For example, period 10 corresponds with time interval $5319\ s - 5910\ s$. The third graph, showing the total number of sent confirmed packets in a simulation period, shows how the algorithm changes this total from 180 (15% of a 1200 device network) to around 60 (5%). The algorithm rapidly reduces the number of sent confirmed packets before settling.

Fig. 4 further explores the impact of the algorithm. The total number of sent confirmed packets reduces (Fig. 3) over time, which causes the total number of retransmissions also to reduce, as seen in Fig. 4. Thanks to the lower number of sent confirmed packets, the number of packets lost due to interference (collisions) reduces. The lower amount of confirmed packets also reduces the number of ACK transmissions required, and thus fewer packets are lost due to ACK transmissions.

The use of the algorithm reduces the effective arrival rate, and subsequently, network performance increases. It is clear that including the earlier intervals in calculations will skew any results as a stable state has not yet been reached. Therefore, the first 20 periods are excluded from the results in this paper.

B. Impact of arrival rate on groupedPackets

The effectiveness of the algorithm across a range of arrival rates is shown in Fig. 5 and Fig. 6 which contain error bars showing the standard deviation. Each simulation consisted of 50 simulation periods with data collected for 30 simulation periods (these periods were after the initial 20 periods as per the previous reasoning). Each node generates one application packet in a simulation period. A simulation period's duration was fixed for each simulation and depended on the targeted arrival rate for a specific simulation. For example, creating an arrival rate of 2.031 pkt/s requires that 1200 nodes transmit every 591 seconds ($1200\ packets/591\ s\ =\ 2.03\ pkt/s$).

For unconfirmed traffic, the algorithm improved in all cases, with the improvement increasing at higher arrival rates. In confirmed traffic's case, the algorithm provided considerable improvements for high arrival rates and a slight reduction (a difference in the mean of ≤ 1 percentage point) at low arrival rates. When comparing standard and groupedPackets, a larger standard deviation can be seen when the algorithm is enabled. This is caused by higher variance in the number of sent confirmed packets, which causes a variance in both performance metrics. In other words, the algorithm's grouping causes some intervals to have more sent confirmed packets than others. Intervals with fewer sent packets tend to have higher packet delivery than intervals with a higher number of sent packets.

C. Interaction of groupedPackets and various base packet sizes.

A packet's size has an impact on its transmission time, and larger packets will require longer ToA, increasing the probability of collisions. The algorithm's ability to aggregate application packets is dependent on the maximum packet size (set to 50 bytes). Fig. 7 and Fig. 8 show the impact of three base packet sizes on the performance. As with previous graphs, an additional random element of between two bytes and eight bytes were still added in all cases. The figures show that performance improves for all ratios and base payloads, with the algorithm having a bigger impact at higher arrival rates.

In Fig. 7, the performance decrease with increased base packet size is similar for both options. The overall reduction in packet delivery as the arrival rate and confirmed ratio increase is evident. The groupedPackets algorithm is more effective for smaller base sizes than for larger ones due to aggregating more packets before reaching the 50-byte limit.

In Fig. 8, variation in base payload size does not affect CPSR if standard LoRaWAN is used, but the use of groupedPackets results in a significant improvement, especially at

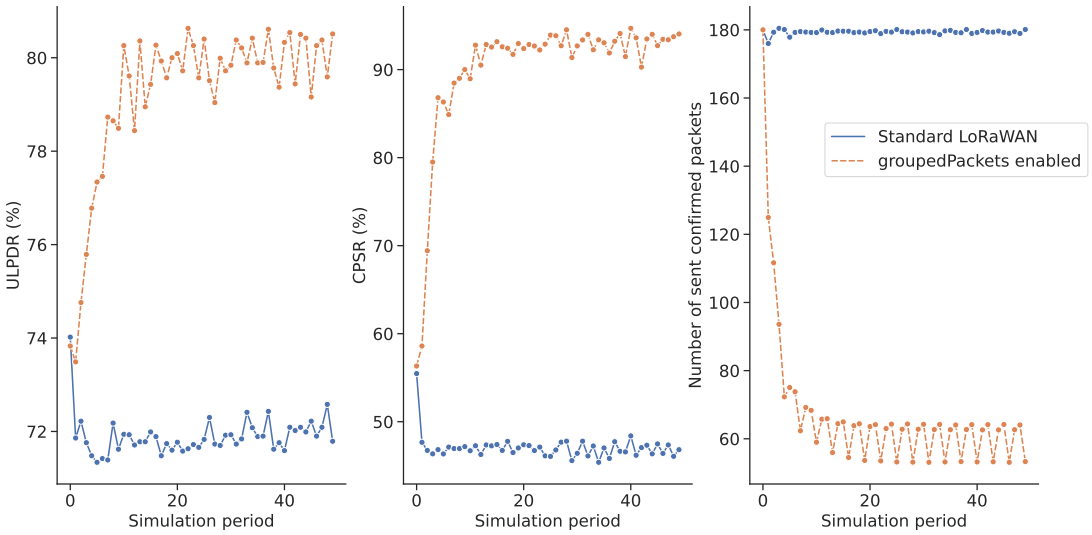


Fig. 3. Performance impact of groupedPackets for a 15 % confirmed network with $\boldsymbol{\lambda = 2.03}$ pkt/s when examined over a long duration.

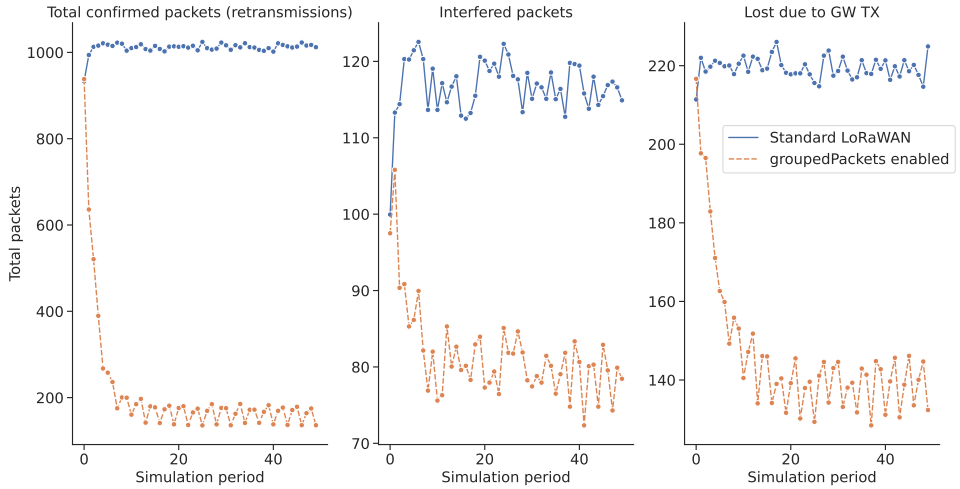


Fig. 4. Analysing the reasons behind the improvement.

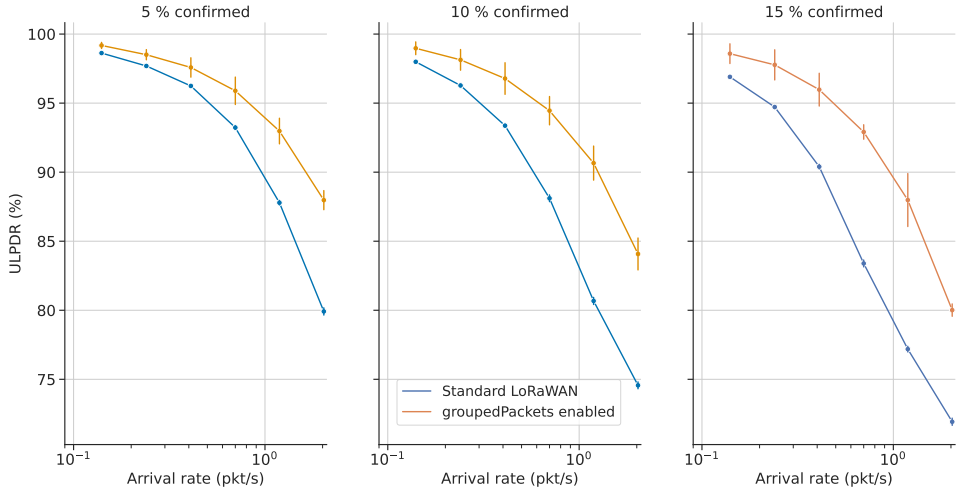


Fig. 5. Performance of unconfirmed traffic with variation in confirmation ratios (30 simulation periods after reaching stable state, base size is 10 bytes).

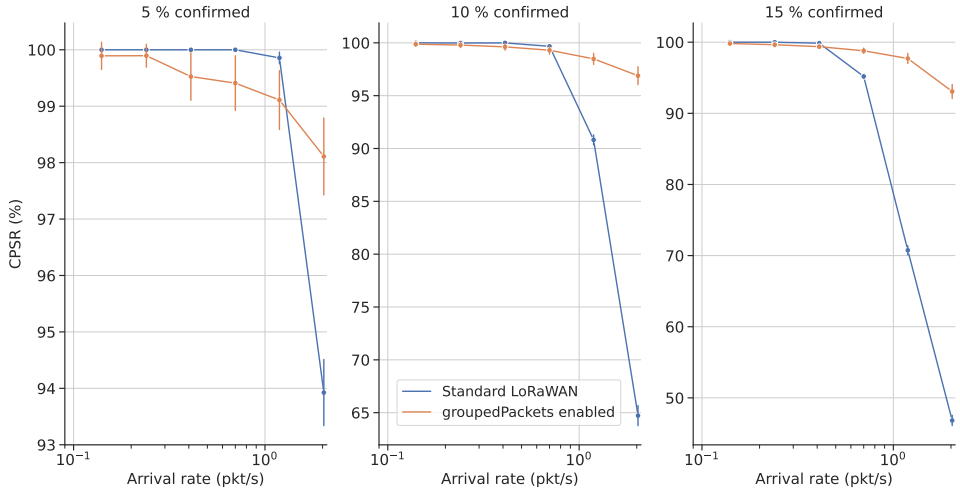


Fig. 6. Performance of confirmed traffic, examined over several arrival rates and confirmed ratios (base size of 10 bytes).

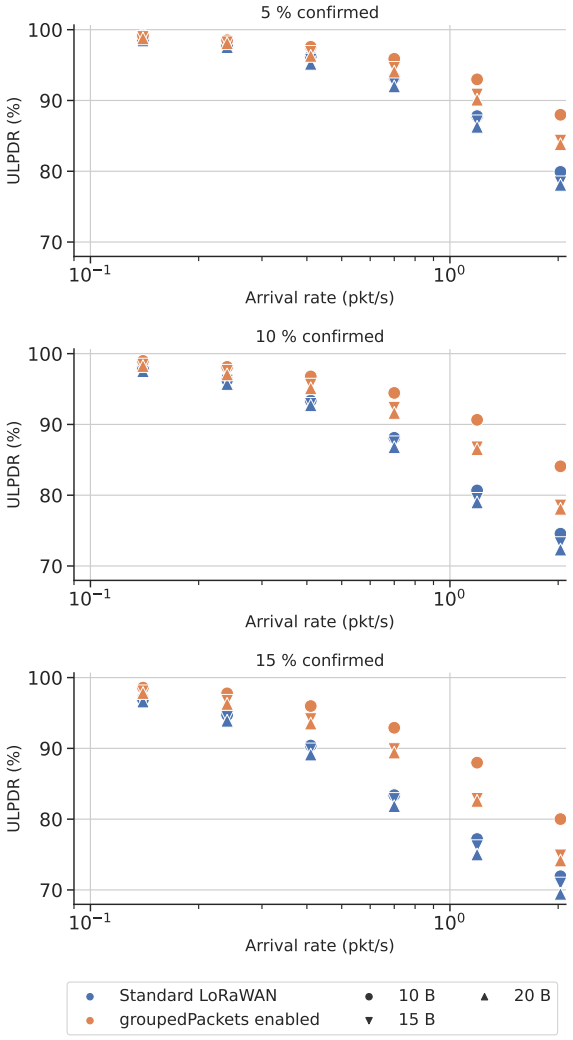


Fig. 7. Performance of unconfirmed traffic, examined over several arrival rates, base application payload sizes and confirmed ratios.

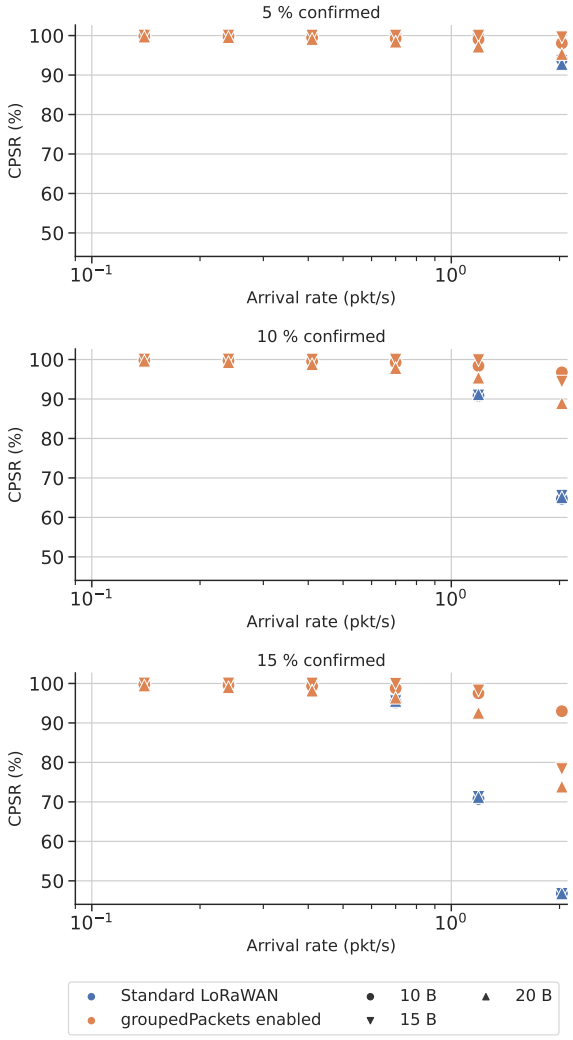


Fig. 8. Performance of confirmed traffic, examined over several arrival rates, base application payload sizes and confirmed ratios.

higher traffic levels. As was the case for unconfirmed traffic, the algorithm is more effective with smaller payloads due to aggregating more payloads. For an arrival rate of 2.03 pkt/s, the algorithm improved the ULPDR of a 15 % confirmed

network by 8.1, 4.0 and 4.7 percentage points for the three base sizes respectively. This improvement is slight, but the CPSR improved by 46.2, 31.6 and 27 percentage points for the same case.

Welch's t-tests were conducted to determine if the mean values for groupedPackets were statistically significant. When the calculated p-values were compared to a significance level of 0.05, only two cases failed to reject the null hypothesis (H_0 = the two group means are identical). These two cases were arrival rates of 0.24 pkt/s and 0.41 pkt/s for a 10 % confirmed network with 15 B base application payloads. In a few cases, t-tests were unable to be conducted as packet success remained at 100 % for both options.

D. Delay introduced by the algorithm

The overall goal of the groupedPackets algorithm is to improve system throughput by improving the probability of successful packet delivery. This improvement requires introducing a time delay for confirmed traffic due to the aggregation of packets. The extent of the delay will vary between nodes as it is a function of DC restrictions, communication distance, SF used, the number of packets being aggregated and the frequency of transmissions.

Fig. 9 shows the average delay between when a node transmits a confirmed packet and an ACK is received in an example network. This graph does not reflect the additional delay experienced by aggregated confirmed payloads but only the delay since transmission of either an aggregated or standard payload. The delay increases at higher arrival rates, as the number of retransmissions required starts to increase, due to increased collisions and GW DC restrictions.

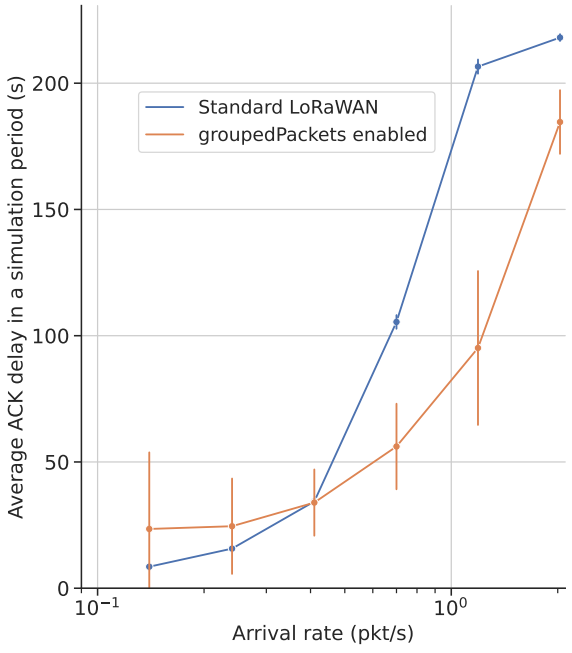


Fig. 9. Average delay to receive an ACK in a 15 % confirmed network sending payloads between 12 B and 18 B.

Networks utilising groupedPackets have a higher standard deviation in their delays which can be attributed to the higher variance in sent confirmed packets (as discussed in Section VII-B). In simulation periods with fewer sent confirmed packets, the gateway can more easily transmit all of the required ACKs whilst remaining within DC restrictions. This results in fewer required retransmissions than in standard networks, which results in nodes receiving ACKs faster. The reverse is true in periods with a higher number of sent packets, and as a result, some variation in the delay is experienced.

Furthermore, this variation in the number of sent confirmed packets start to become beneficial for groupedPackets at higher arrival rates, as it reduces the number of retransmissions required. Whilst groupedPackets appears to offer a lower average delay than standard networks, one must remember to add on the additional delay caused by the aggregation of payloads.

For example, if a node aggregates 4 packets and would have normally transmitted a packet every 5 minutes, these packets would have respectively experienced an additional 15 minutes, 10 minutes, 5 minutes and 0 minutes delay before transmission. Applications sensitive to the end-to-end delay between the NS and a node can opt out of using groupedPackets or ignore aggregation requests if a specific event requires an immediate response. LoRaWAN class A is not optimised for latency, and one of the other device classes or another LPWAN technology should be used for delay-sensitive applications.

VIII. Potential Deployment Scenarios

There are several IoT use cases where the deployment of ACS and the groupedPackets algorithm can be of benefit. An example would be cold chain control, in which IoT devices are used to monitor the temperature of products such as vaccines and produce. An extensive collection of these monitoring devices would lead to congestion, such as Internet of Ships scenarios where intelligent containers are used to monitor goods such as food, medicine and chemicals during travel or storage in warehouses [35].

The algorithm can also be used with minor modifications in scenarios where congestion is caused by temporary clustering of devices. This type of scenario is typical of Smart Agriculture, such as where devices are used to monitor cattle [36]. In these situations, congestion is increased for a short duration, but the aggregation can be reduced once devices are more spread out. An example is a free range smart dairy farm with multiple gateways, while the milking parlour has only one gateway. No aggregation would be required whilst cows are grazing, but when bundled up at the parlour, they can quickly overwhelm a single gateway.

Finally, LoRaWANs have shown promise for deployment in emergency scenarios or natural disasters [37]. In emergencies, LoRaWANs provide a way to provide coverage for many devices with minimal resources. Utilising groupedPackets can allow a single gateway to better function in situations where multiple gateways cannot be deployed due to constraints such as power or time.

IX. Conclusion

This paper investigated congestion caused in LoRaWANs by downlink traffic. Simulations showed that in large networks, performance is severely impacted and influenced by packet arrival rates and the ratio between confirmed and unconfirmed traffic. The ACS was developed to help counteract this problem, which monitors a network for congestion and activates a created algorithm called groupedPackets to reduce congestion. This is done by requesting that confirmed nodes aggregate their application data packets. Simulations showed that this effectively improved both ULPDR and CPSR in congested LoRaWANs. In the future, this algorithm could be expanded to also consider unconfirmed nodes and additional algorithms developed to expand the ACS.

References

[1] Y. Song, F. R. Yu, L. Zhou, X. Yang, and Z. He, "Applications of the Internet of Things (IoT) in Smart Logistics: A Comprehensive Survey," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4250–4274, 2021.

[2] M. A. Albreem, A. M. Sheikh, M. H. Alsharif, M. Jusoh, and M. N. Mohd Yasin, "Green Internet of Things (GIoT): Applications, Practices, Awareness, and Challenges," *IEEE Access*, vol. 9, pp. 38 833–38 858, 2021.

[3] L. Feltrin, C. Buratti, E. Vinciarelli, R. De Bonis, and R. Verdone, "LoRaWAN: Evaluation of link- and system-level performance," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2249–2258, 2018.

[4] E. M. Torroglosa-Garcia, J. M. Calero, J. B. Bernabe, and A. Skarmeta, "Enabling Roaming across Heterogeneous IoT Wireless Networks: LoRaWAN MEETS 5G," *IEEE Access*, vol. 8, pp. 103 164–103 180, 2020.

[5] F. Van Den Abeele, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Scalability analysis of large-scale LoRaWAN networks in ns-3," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2186–2198, 2017.

[6] J. M. Marais, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on the viability of confirmed traffic in a LoRaWAN," *IEEE Access*, vol. 8, pp. 9296–9311, 2020.

[7] M. Bagaa, Y. Challal, A. Ksentini, A. Derhab, and N. Badache, "Data Aggregation Scheduling Algorithms in Wireless Sensor Networks: Solutions and Challenges," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1339–1368, 2014.

[8] 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, no. 1, pp. 669–683, 2020.

[9] Y. Hasegawa and K. Suzuki, "A Multi-User ACK-Aggregation Method for Large-Scale Reliable LoRaWAN Service," in *IEEE International Conference on Communications (ICC)*, Shanghai, China, May 2019, pp. 1–7.

[10] J. Lee, W. C. Jeong, and B. C. Choi, "A Scheduling Algorithm for Improving Scalability of LoRaWAN," in *International Conference on Information and Communication Technology Convergence, ICTC 2018*, Jeju, South Korea, Oct. 2018, pp. 1383–1388.

[11] J. Lee, Y. S. Yoon, H. W. Oh, and K. R. Park, "DG-LoRa: Deterministic Group Acknowledgment Transmissions in LoRa Networks for Industrial IoT Applications," *Sensors*, vol. 21, no. 4, pp. 1–18, 2021.

[12] C. Trasviña-Moreno, R. Blasco, Á. Marco, R. Casas, and A. Trasviña-Castro, "Unmanned Aerial Vehicle Based Wireless Sensor Network for Marine-Coastal Environment Monitoring," *Sensors*, vol. 17, no. 3, p. 460, 2017.

[13] D. Croce, M. Gucciardo, S. Mangione, G. Santaromita, and I. Tinnirello, "Impact of LoRa Imperfect Orthogonality: Analysis of Link-level Performance," *IEEE Communications Letters*, vol. 22, no. 4, pp. 796–799, 2018.

[14] D. Magrin, M. Centenaro, and L. Vangelista, "Performance Evaluation of LoRa Networks in a Smart City scenario," in *IEEE International Conference on Communications (ICC)*, Paris, France, Apr. 2017, pp. 1–7.

[15] LoRa Alliance, "LoRaWAN 1.1 Specification," p. 101, 2017.

[16] ——, "LoRaWAN 1.1 Regional Parameters," *LoRa Alliance*, pp. 1–72, 2018.

[17] R. Kufakunesu, G. P. Hancke, and A. M. Abu-Mahfouz, "A Survey on Adaptive Data Rate Optimization in LoRaWAN: Recent Solutions and Major Challenges," *Sensors (Switzerland)*, vol. 20, no. 18, pp. 1–25, 2020.

[18] J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Low overhead scheduling of LoRa transmissions for improved scalability," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3097–3109, 2019.

[19] B. Kim and K. I. Hwang, "Cooperative Downlink Listening for Low-Power Long-Range Wide-Area Network," *Sustainability*, vol. 9, no. 4, pp. 1–15, 2017.

[20] M. Centenaro and L. Vangelista, "Time-Power Multiplexing for LoRa-Based IoT Networks: An Effective Way to Boost LoRaWAN Network Capacity," *International Journal of Wireless Information Networks*, pp. 1–11, 2019. [Online]. Available: https://doi.org/10.1007/s10776-019-00437-8

[21] D. Zorbas and X. Fafoutis, "Time-Slotted LoRa Networks: Design Considerations, Implementations, and Perspectives," *IEEE Internet of Things Magazine*, pp. 84–89, Mar. 2021.

[22] S. Dawaliby, A. Bradai, and Y. Pousset, "Adaptive dynamic network slicing in LoRa networks," *Future Generation Computer Systems*, vol. 98, pp. 697–707, Apr. 2019.

[23] T. Deng, J. Zhu, and Z. Nie, "An Improved LoRaWAN protocol based on Adaptive Duty Cycle," in *2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference, ITOEC 2017*, Chongqing, China, 2017, pp. 1122–1125.

[24] D. Magrin, "Network level performances of a LoRa system," Master's thesis, Università di Padova, Padua, Italy, 2016.

[25] M. Capuzzo, D. Magrin, and A. Zanella, "Confirmed traffic in LoRaWAN: Pitfalls and countermeasures," in *17th Annual Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2018*, Capri, Italy, June 2018, pp. 1–7.

[26] D. Magrin, M. Capuzzo, and A. Zanella, "A Thorough Study of LoRaWAN Performance Under Different Parameter Settings," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 1–12, 2020.

[27] N. Blenn and F. Kuipers, "LoRaWAN in the wild: Measurements from the things network," *arXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03086

[28] D. Magrin, D. Zhou, and M. Zorzi, "A simulation execution manager for ns-3 encouraging reproducibility and simplifying statistical analysis of ns-3 simulations," in *MSWiM 2019 - Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Nov. 2019, pp. 121–125.

[29] B. Reynders, Q. Wang, P. Tuset-Peiro, X. Vilajosana, and S. Pollin, "Improving reliability and scalability of LoRaWANs through lightweight scheduling," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1830–1842, 2018.

[30] A. I. Pop, U. Raza, P. Kulkarni, and M. Sooriyabandara, "Does Bidirectional Traffic Do More Harm Than Good in LoRaWAN Based LPWA Networks?" in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Singapore, Singapore, Dec. 2017, pp. 1–6. [Online]. Available: http://arxiv.org/abs/1704.04174

[31] V. Di Vincenzo, M. Heusse, and B. Tourancheau, "Improving Downlink Scalability in LoRaWAN," in *IEEE International Conference on Communications*. Shanghai, China, China: IEEE, May 2019, pp. 1–7.

[32] Multi-Tech Systems, "MultiTech Conduit," https://www.multitech.com/brands/multiconnect-conduit (Accessed: 20 Oct. 2022).

[33] Arm Limited, "MultiTech mDot," https://os.mbed.com/platforms/MTS-mDot-F411/ (Accessed: 20 Oct. 2022).

[34] STMicroelectronics, "STM32WLE5JC," https://www.st.com/en/microcontrollers-microprocessors/stm32wle5jc.html (Accessed: 20 Oct. 2022).

[35] S. Aslam, M. P. Michaelides, and H. Herodotou, "Internet of ships: A survey on architectures, emerging applications, and challenges," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9714–9727, 2020.

[36] D. Heeger, M. Garigan, E. E. Tsiropoulou, and J. Plusquellic, "Secure Energy Constrained LoRa Mesh Network," in *Ad-Hoc, Mobile, and Wireless Networks*, Bari, Italy, Oct. 2020, pp. 228–240.

[37] J. Navarro-Ortiz, J. J. Ramos-Munoz, J. M. Lopez-Soler, C. Cervello-Pastor, and M. Catalan, "A LoRaWAN Testbed Design for Supporting Critical Situations: Prototype and Evaluation," *Wireless Communications and Mobile Computing*, pp. 1–13, 2019.

Jaco Morné Marais (S'17) received the BEng and MEng from the University of Pretoria in 2015 and 2018 respectively and is currently completing his PhD at the University of Pretoria. His research interests include LoRaWAN, low power wide area networks, wireless sensor networks and embedded systems.

Adnan M. Abu-Mahfouz (M'12—SM'17) received his MEng and PhD degrees in computer engineering from the University of Pretoria. He is currently a Chief Researcher and the Centre Manager of the Emerging Digital Technologies for 4IR (EDT4IR) research centre at the Council for Scientific and Industrial Research (CSIR), Extraordinary Professor at University of Pretoria, Professor Extraordinaire at Tshwane University of Technology and Visiting Professor at University of Johannesburg. His research interests are wireless sensor and actuator network, low power wide area networks, software defined wireless sensor network, cognitive radio, network security, network management, sensor/actuator node development. He is a Section Editor-in-Chief at the Journal of Sensor and Actuator Networks, an associate editor at IEEE Access, IEEE Internet of Things and IEEE Transaction on Industrial Informatics, Senior Member of the IEEE and Member of many IEEE Technical Communities.

Gerhard P. Hancke (M'88-SM'00-F'16-LF'19) received the B.Sc. and B.Eng. degrees (1970), and the M.Eng. degree (1973) in Electronic Engineering from the University of Stellenbosch, South Africa, and the D.Eng. degree (1983) from the University of Pretoria, South Africa. He is a Professor with the Nanjing University of Posts and Telecommunications, China and the University of Pretoria, South Africa. He is recognized internationally as a pioneer and leading scholar in Industrial Wireless Sensor Networks research. He initiated and co-edited the first Special Section on Industrial Wireless Sensor Networks in the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS in 2009 and the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS in 2013. He co-edited a textbook, Industrial Wireless Sensor Networks: Applications, Protocols and Standards (2013), the first on the topic. Prof. Hancke has been serving as an Associate Editor and Guest Editor for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE ACCESS, and the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS. Currently, he is a Co-Editor-in-Chief of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS and Senior Editor of IEEE ACCESS.