Improving LoRa Scalability by a Recursive Reuse of Demodulators

Alexandre Guitton, Megumi Kaneko

Improving LoRa Scalability by a Recursive Reuse of Demodulators

Alexandre Guitton[(1)], Megumi Kaneko[(2)]
(1) Université Clermont Auvergne, CNRS, LIMOS, F-63000 Clermont-Ferrand, France
(2) National Institute of Informatics, Hitotsubashi, 2-1-2, Chiyoda-ku, 101-8430 Tokyo, Japan
Emails: alexandre.guitton@uca.fr, megkaneko@nii.ac.jp

***Abstract*—Long Range (LoRa) is a protocol that enables low-power wireless communications over long distances for a wide range of IoT applications. Its main drawback is its limited throughput, which is further reduced by the small number of demodulators in the hardware of the gateway. In this paper, we propose a first algorithm that allows demodulators to be reused for short frames while waiting for the end of the preamble of long frames, and a second that further plans the demodulation of future frames while demodulating. By smartly planning the demodulation of multiple frames, the proposed methods enable a recursive reuse of each demodulator. Compared to the benchmark packet arbiter policy, our methods are shown to offer throughput and fairness enhancements even with a large number of users, thereby improving the scalability of LoRa systems.**

I. Introduction

Internet of Things (IoT) systems will be key enablers of the upcoming 5G and Beyond and 6G applications. In that context, Long Range (LoRa) and LoRa Wide Area Network (LoRaWAN) are gathering more and more research interests. LoRa [1] is a physical layer protocol that enables wireless communication over long distances, designed to support many IoT devices with a low-cost infrastructure. LoRaWAN [2] is the main MAC protocol on top of LoRa. It enables end-devices to communicate to gateways with low energy consumption. However, the main drawback of both LoRa and LoRaWAN is their low throughput. Indeed, their bitrate was intentionally limited by LoRa designers so as to increase the robustness of the modulation, and hence the communication range. Moreover, only 1% of this bitrate can be used in most cases in order to enforce European regulations on the limited duty-cycle. LoRaWAN also introduces additional overhead to deal with collisions, further reducing the achievable throughput.

The hardware of LoRaWAN gateways enables the demodulation of several signals in parallel, when they are received with different parameters such as channel frequency or spreading factor (SF). For that, the gateway chip integrates several demodulators, typically eight, each capable of demodulating a single frame at a given time [3]. The packet arbiter microcontroller unit (MCU) is in charge of configuring the demodulators on the fly. However, the number of demodulators drastically limits the number of frames that can be demodulated in parallel, and thus constitutes another key factor that reduces the throughput of LoRaWAN [4]. This imposes a strong limitation on the number of supported end-devices, which is a major shortcoming of LoRa-based systems as they are expected to support an exponential increase of IoT mobile data traffic. Nevertheless, this problem had been overlooked in most references in the literature. Among the few exceptions, [4] proposed a mathematical model that includes the number of demodulators as a parameter. They show that this limitation significantly reduces the throughput, and thus suggest that future theoretical studies have to integrate the number of demodulators. The limitation of the number of demodulators was also considered in [5] as a practical hardware limitation for the network capacity. This unavoidable hardware constraint needs to be properly integrated in order to assess and improve the scalability of LoRa and LoRaWAN systems.

In this paper, we propose a method for alleviating this hardware constraint, by enabling the reuse of each demodulator by multiple frames in parallel, a process hereafter referred to as recursive reuse. More precisely, we decouple the actual demodulation performed by the demodulator, from the planning of the demodulation performed by the packet arbiter MCU. Currently, when a preamble is detected, an idle demodulator is booked for this frame until the end of the frame's payload. By contrast, in our proposal, we use the following two properties: (i) When a demodulator is booked to demodulate a frame but is still waiting for the beginning of its payload, the demodulator can be used to demodulate other frames, provided that they are short enough. (ii) When a demodulator is busy demodulating the payload of a frame, the demodulator can be booked for other frames, provided that their payload starts after the end of the current payload.

The contributions of this work are threefold.
(1) We propose an algorithm called FIFO-RR1 that enables a LoRa demodulator to be recursively reused to demodulate short frames while it is waiting for the end of a preamble.
(2) We propose another algorithm called FIFO-RR2 that plans the demodulation of upcoming payloads while the demodulator is busy demodulating a given payload.
(3) Computer simulation results show that, compared to the benchmark packet arbiter policy, our algorithms enhance the system throughput and fairness. These gains are maintained even for a large number of end-devices, thereby providing significant scalability improvements.

II. System model

In this section, we first describe the LoRa and LoRaWAN protocols, followed by the hardware architecture of the

SX1301 chip, which is used by LoRaWAN gateways.

A. LoRa and LoRaWAN

LoRa [1] is a physical layer protocol designed to enable low-power long range communications, at the cost of a reduced bitrate. It uses a chirp spread spectrum modulation where symbols are encoded as linear frequency sweeps. Symbols can be either upchirps if their frequency increases, or downchirps otherwise. The duration of each symbol depends on SF: increasing SF reduces the bitrate by increasing the symbol duration, but increases the robustness and thus the communication range. Communications on different SFs are quasi-orthogonal, and can often be demodulated in parallel.

LoRa frames start with a preamble followed by the data payload. The preamble consists of a series of repeated upchirps (eight for EU863-870), two upchirps for the sync word (equal to 0x34 for LoRaWAN), and two and a quarter downchirps for the end of the preamble. The data payload is a series of upchirps, where each symbol carries a value among 2^{SF}.

LoRaWAN [2] is a MAC protocol which works on top of LoRa, with variations called regional settings, based on national regulations. Here, we focus on the EU863-870 regional settings. To transmit a frame to a gateway, an end-device uses a random channel and transmits without channel sensing. After the transmission, the end-device opens two receive windows for possible acknowledgments. The end-device cannot send another frame on the same subband until a long time-off period, defined as 99 times the duration of the frame, in the case of 1% duty-cycle. Channels in LoRaWAN are orthogonal, thus transmissions can be received in parallel. LoRaWAN defines several datarates, corresponding to different SFs.

B. Packet arbiter of the SX1301 chip

Figure 1 shows the architecture of the receiver part of the SX1301 chip. When the SX1301 receives a radio signal, it is filtered by the ten programmable reception paths (denoted IF0 to IF9) in parallel, based on the channel, the bandwidth and the datarate. IF0 to IF7 scan for possible preambles at all time. When a new preamble is detected on these eight reception paths, the information is passed to the packet arbiter, which decides whether or not to allocate a LoRa demodulator for this specific frame. IF8 and IF9 demodulate the filtered signal using respectively LoRa and GFSK demodulation schemes.

The reception paths have different characteristics. IF0 to IF7 need to be preconfigured with a given frequency, have a bandwidth of 125 kHz. They are rate-independent, *i.e.*, they are able to detect preambles on any datarate. IF8 needs to be preconfigured with a given frequency, bandwidth, and datarate, and is used for backhaul LoRa links to other gateways. IF9 enables to demodulate high datarate GFSK signals. Hereafter, we will omit IF8 and IF9 as we focus on LoRa uplinks.

If IF0 to IF7 are configured to scan different channels, the packet arbiter will receive the preambles for all SFs on all these channels, simultaneously. Since there are six possible SFs in LoRaWAN (from SF7 to SF12), the chip emulates 48 LoRa demodulators. However, only eight frames at most can be demodulated, one for each demodulator.

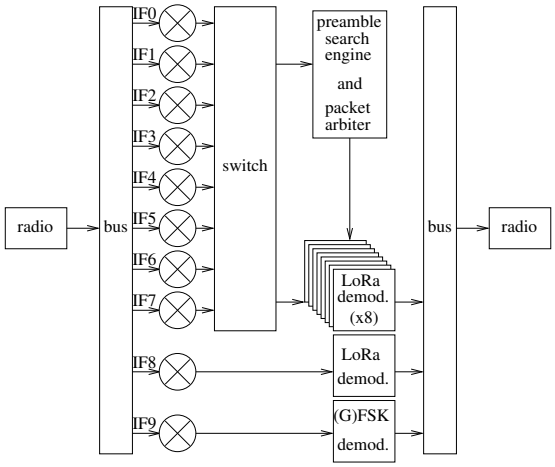

Fig. 1. Architecture of the receiver part of the SX1301 chip.

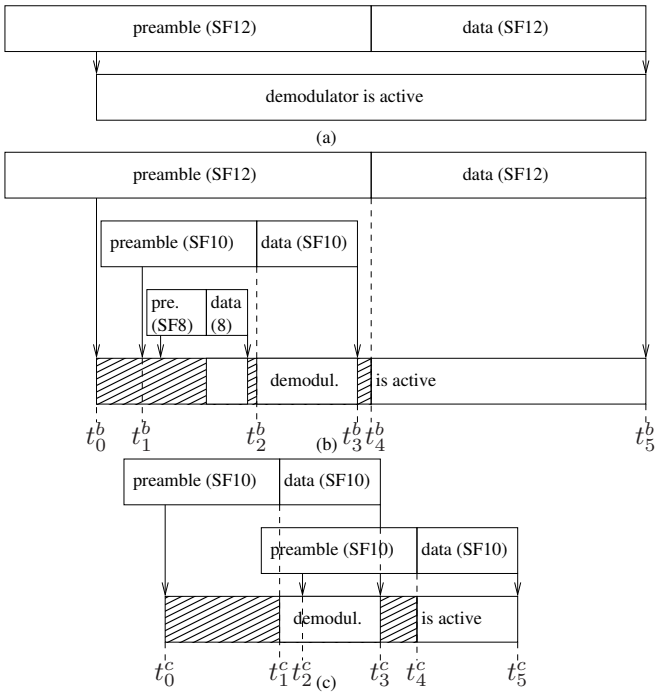

Fig. 2. (a) A typical packet arbiter allocates a demodulator as soon as a preamble is detected. (b) A demodulator can be (recursively) reused during the preamble of a frame with a large SF, to demodulate frames with small SFs. (c) While a demodulator is busy, it can still be booked for future demodulations.

The default policy implemented in the packet arbiter MCU is not given in the datasheet of SX1301 [3]. Hence, we assume reasonably that it is a simple FIFO policy: as soon as a new preamble is detected, an available demodulator (if any) is allocated to demodulate the frame.

III. Proposed method

In this section, we present how a demodulator can be reused for short frames during the end of the preamble of long frames, together with the first proposed packet arbiter policy FIFO-RR1. Then, we present how a demodulator can be booked for

a future frame payload while it is demodulating the payload of another frame, together with the second proposed packet arbiter policy FIFO-RR2.

A. Recursive reuse of demodulators and FIFO-RR1

The limited number of demodulators (typically, eight) limits the throughput achievable by a LoRaWAN gateway, as it is not possible to simultaneously demodulate a larger number of frames than of demodulators. Thus, our strategy is to identify the time intervals during which a demodulator is booked but not actively used, and to reuse it for short frames.

The demodulator's role is to extract the values of the data symbols from the payload. Thus, the demodulator is only useful during the payload. However, with the current packet arbiter policy, a demodulator is allocated to a frame as soon as a new preamble is detected, as shown in Fig. 2(a). We thus propose to use the duration between the detection of the preamble and the end of the preamble to demodulate short frames. Recursively, during the preambles of these short frames, the demodulator can be reused for shorter frames.

Figure 2(b) gives an example of the proposed recursive reuse of a demodulator. Let us assume that a first frame is sent with SF12. At t_0^b, after a few symbols of the preamble, the preamble is detected. The packet arbiter MCU books the demodulator for this first frame: as soon as the end of the preamble will be detected (that is, at t_4^b), the demodulator will start demodulating this frame. In the meantime, however, it is available for short frames. A second frame arrives with SF10 while the demodulator is booked. When the preamble of this second frame is detected at t_1^b, the packet arbiter MCU detects that the demodulator is currently not used (although it is booked), and that it could demodulate the whole frame before the start of the payload of the first frame, because $t_3^b < t_4^b$. Thus, the packet arbiter MCU books the demodulator for this second frame. Then, a third frame arrives with SF8. Again, the packet arbiter MCU detects that this third frame can be demodulated before the payload of the second frame starts (that is, before t_2^b). Thus, this third frame is booked. Shortly after, the third frame is demodulated, followed by the second frame during $[t_2^b; t_3^b]$, and finally by the first frame during $[t_4^b; t_5^b]$. Note that it is possible to demodulate several short frames in sequence, *e.g.*, it is possible to demodulate seven SF7 frames of 8 bytes (36.10 ms per frame) during the reuse window of a SF12 frame (270.35 ms).

The proposed operations require four variables: (i) the variable $state[d]$ stores the state of each demodulator d, among `IDLE`, `BOOKED` and `BUSY`, (ii) the variable $next[d]$ stores parameters of the next frame for d, (iii) the variable $timeStack[d]$ is a stack of payload times for d, and (iv) the variable $frameStack[d]$ is a stack of frame parameters for d.

Algorithm 1 describes the proposed operations when the packet arbiter MCU detects a new preamble on SF s. First, the packet arbiter searches for an available demodulator, or a demodulator which is booked for a later demodulation (provided that the demodulation of the previous frame starts after the demodulation of the current frame). If such a demodulator is found, the demodulator is booked and a new timer is setup to expire at the end of the preamble. $timeStack$ is used to store the times at which a demodulator is planned to become busy. $frameStack$ is used to store the parameters of the frame, such as the SF s and the channel. For instance, on the example of Figure 2(b), let us first consider that $t_{current} = t_0^b$. As a new preamble is detected, the algorithm detects that the demodulator d is `IDLE`, so the demodulator d becomes `BOOKED` until $next[d] = t_4^b$, and a timer will expire at t_4^b. Shortly after, at $t_{current} = t_1^b$, a new preamble is detected. The demodulator d is `BOOKED` and $next[d] = t_4^b > t_3^b$, so d has enough time to demodulate this new frame before the payload of the previous frame starts. Thus, this second frame with SF10 is booked for demodulation.

Algorithm 1: Demodulator allocation upon the preamble detection, for FIFO-RR1.

input: a new preamble is detected on SF s at $t_{current}$
$t_{max}(s) \leftarrow$ remaining time on air for the longest frame at SF s
if *there is* d *such that* $state[d] =$ `IDLE` **or** ($state[d] =$ `BOOKED` *and* $next[d] > t_{current} + t_{max}$) **then**
  $t_{preamble} \leftarrow$remaining preamble duration for SF s
  $next[d] \leftarrow t_{current} + t_{preamble}$
  push $next[d]$ on $timeStack[d]$
  push the frame parameters on $frameStack[d]$
  $state[d] \leftarrow$ `BOOKED`
  $demodulator[d] \leftarrow$ parameters of the frame
  start timer for $t_{preamble}$
else
  frame is rejected
end

The actual demodulation starts when the timer expires, or when the end of a preamble is detected before the timer expired in case of an incorrect estimation of the preamble duration. The demodulator state then becomes `BUSY`.

Algorithm 2 is used when a demodulator finishes demodulating a frame. Depending on the stacks, the demodulator becomes either `IDLE` if there is no reservation or `BOOKED`.

Algorithm 2: Demodulator reuse after the payload.

input: A demodulator d finishes demodulating a frame
pop $timeStack[d]$
pop $frameStack[d]$
if $timeStack[d]$ *is empty* **then**
  $state[d] \leftarrow$ `IDLE`
  $demodulator[d] \leftarrow \emptyset$
else
  $state[d] \leftarrow$ `BOOKED`
  $next[d] \leftarrow$ top of $timeStack[d]$
  $demodulator[d] \leftarrow$ top of $frameStack[d]$
  start timer in $next[d] - t_{current}$
end

The proposed FIFO-RR1 policy consists in implementing Algorithm 1 and Algorithm 2. Note that Algorithm 1 takes into account the max frame duration t_{max}. However, this duration is typically very long. We propose to reduce the maximum duration depending on the application. Note that it is always possible to assume that a frame is short and to cancel the demodulation if the frame size (at the beginning of the header), is too large or if the frame duration is too long.

Implementing these algorithms increases the complexity of the packet arbiter MCU. However, they are simple enough to be implemented efficiently. Moreover, the memory overhead of these three algorithms is limited. For each demodulator, there is a $state$ variable with three possible values (2 bits), a $next$ variable for the next demodulation time (16 bits), a $timeStack$ variable of 16 bits per item, and a $frameStack$ variable of one byte per item (including both SF and channel). According Table I, which represents the reuse window duration as well as the maximum time on air (TOA) of frames for each SF, the number of items in the stack is at most three. Indeed, an SF7 frame of 8 bytes (36.10 ms) can be demodulated during the reuse window (67.56 ms) of an SF10 frame of 8 bytes (247.81 ms), which can be demodulated during the reuse window (270.35 ms) of an SF12 frame. Overall, the memory overhead is of at most 90 bits per demodulator.

SF	Reuse window	TOA (max)	TOA (20b)	TOA (8b)
12	270.35 ms	2465.79 ms	1318.91 ms	991.23 ms
11	135.13 ms	1314.82 ms	741.38 ms	495.62 ms
10	67.56 ms	616.45 ms	370.69 ms	247.81 ms
9	33.82 ms	615.42 ms	185.34 ms	123.90 ms
8	16.91 ms	614.91 ms	102.91 ms	72.19 ms
7	8.41 ms	348.42 ms	56.58 ms	36.10 ms

TABLE I
TIME DURING WHICH A DEMODULATOR CAN BE REUSED, FOR EACH SF.

B. Booking of busy demodulators and FIFO-RR2

As shown in Algorithm 1, FIFO-RR1 does not consider demodulators that are in the state `BUSY`, as they are already busy demodulating frames. However, only preambles detected after the end of the demodulation are considered. This causes a wasted time between the demodulation of two frames arriving in sequence, of about the duration of a preamble. Let us consider the example of Figure 2(c). If FIFO-RR2 is not used, the second frame whose preamble is detected at t_2^c would be dropped by the demodulator. The demodulator would have to wait to detect a preamble after t_3^c, and then the end of this preamble, until it can actually demodulate the frame.

We propose to consider busy demodulators when new preambles are detected, as long as the beginning of the payload of the new frame starts after the end of the demodulation of the current frame. In the example of Figure 2(c), the second frame can be booked as its payload starts at t_4^c, which is after the end of the payload of the current frame at t_3^c.

Algorithm 3 provides the demodulator allocation operations for the proposed FIFO-RR2 algorithm, while Algorithm 2 is kept unchanged. The priority is still to allocate demodulators that are either `IDLE` or `BOOKED` using Algorithm 1. Now, however, a demodulator is also considered available if the demodulation of the current payload finishes before the end of the detected preamble. In this case, the detected frame is inserted in the stack after the top frame (which is the frame currently processed). For simplicity reasons, we considered only the case where the stack size is equal to one, that is, there is no ongoing recursion. Otherwise, we would have to check that the new frame does not overlap other initially planned demodulations by checking the whole stack. For instance, on the example of Figure 2(c), let us consider that $t_{current} = t_2^c$. At this time, d is demodulating the first frame and is thus `BUSY`. When detecting the preamble of the second frame, the packet arbiter detects that d will finish demodulating the current frame (at t_3^c) before the payload of the second frame starts (at t_4^c). Thus, it is possible to book the second frame after the first frame. In order to reuse the previous algorithms, this requires the second frame to be below the first frame in the stacks.

Algorithm 3: Demodulator allocation after the preamble detection, for FIFO-RR2.

input: a new preamble is detected on SF s at $t_{current}$
execute Algorithm 1
if *the frame was rejected by Algorithm 1* **then**
  $t_{preamble} \leftarrow$ remaining duration for the new preamble
  $t_{newPayload} \leftarrow t_{current} + t_{preamble}$
  `/* compute the time when each busy demodulator finishes its frame */`
  forall *demodulator* d *such that* $state[d] =$ `BUSY` **do**
    $payloadDuration \leftarrow$payload duration of frame on top of $frameStack[d]$
    $t_{end}[d] \leftarrow next[d] + payloadDuration$
  end
  if *there is* d *such that* $state[d] =$ `BUSY` *and* $t_{end}[d] \leq t_{newPayload}$ *and size of* $frameStack[d]$ *equals to 1* **then**
    `/* insert the new time in the stack */`
    $previousTopTime \leftarrow$ pop $timeStack[d]$
    push $t_{newPayload}$ on $timeStack[d]$
    push $previousTopTime$ on $timeStack[d]$
    `/* insert the new frame in the stack */`
    $previousTopFrame \leftarrow$ pop $frameStack[d]$
    push the new parameters on $frameStack[d]$
    push $previousTopFrame$ on $frameStack[d]$
  else
    frame is rejected
  end
end

IV. SIMULATION RESULTS

In order to evaluate the performance of our packet arbiter policies, we considered two baseline policies: the MAX policy and the FIFO policy. The MAX policy ideally assumes an

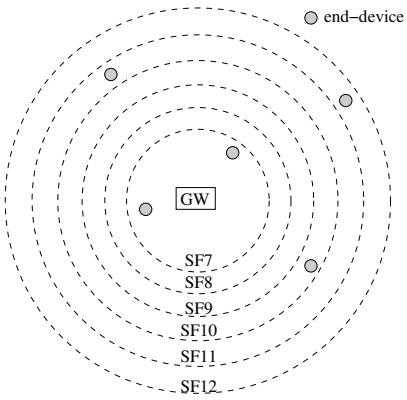

Fig. 3. An example of a deployment around a single gateway.

infinite number of demodulators, and is thus impractical. The FIFO policy (see Subsection II-B) is the benchmark policy.

A. Metrics and simulation parameters

Our first metric is the overall throughput, computed as the number of demodulated frames. We denote by $n_P(sf)$ the number of frames of SF sf demodulated by policy P. Note that $n_{MAX}(sf)$ is equal to the number of transmitted frames for SF s. Our second metric is the fairness among SFs. Since SF7 frames are much shorter than SF12 frames, a packet arbiter policy prioritizing SF7 frames would increase the throughput compared to a fair policy, but that would penalize end-devices that have to use a larger SF. We compute the fairness $f(P)$ of a policy P by using Jain's fairness index: $f(P) = \frac{A(P)^2}{B(P)}$, where $A(P) = \sum_{sf=7}^{12} n_P(sf)/n_{MAX}(sf)$ and $B(P) = \sum_{sf=7}^{12}((n_P(sf)/n_{MAX}(sf))^2)$. Maximum fairness is achieved at 1.

We make the following assumptions:
(i) We consider that it takes 4 symbols to detect the preamble. The exact preamble detection duration is unknown, but our assumption is in line with findings from other authors. In [6], the authors performed extensive simulations to characterize collisions. They identified that the receiver locking time was 4 symbols. Note that the preamble detection might be smaller than this receiver locking time. In [7], the authors studied the interference in LoRa and found out that the receiver needs only six symbols from the preamble to be synchronized with the transmitter. They also mention that the gateway needs few symbols to detect the start of a frame.
(ii) We consider a perfect channel and we assume that there is no collision among frames.

Our results are averaged over 100 runs. In each run, frames are generated for 10000 seconds, and each node has a duty-cycle of 1%. Nodes are randomly deployed around a single gateway, as shown on Figure 3. They use a SF which depends on their distance to the gateway. The proportion of nodes for each SF is a function of the area size covered by each SF according to the sensitivity threshold, and is given by: 21% of nodes use SF7, 8% use SF8, 12% use SF9, 17% use SF10, 19% use SF11, and 23% use SF12 (see [8], [9]).

B. Effects of the number of nodes

Figure 4 shows the throughput performance (number of demodulated frames) against the number of nodes, for the four packet arbiter policies, and for eight demodulators. The throughput of the baseline MAX policy increases linearly with the number of nodes, as the number of generated frames increases with the number of nodes. The throughput of the baseline FIFO policy increases with the number of nodes, as there are more opportunities to demodulate frames. However, the percentage of demodulated frames varies between 91% for 100 nodes to 58% for 1000 nodes. FIFO-RR1 and FIFO-RR2 have a similar performance. These two algorithms allow to demodulate more frames than FIFO: the gain of FIFO-RR1 compared to FIFO varies between 1.24% for 100 nodes to 7.62% for 1000 nodes, while the gain of FIFO-RR2 compared to FIFO varies between 5.88% for 100 nodes to 8.09% for 250 nodes, and then remains at about 6.5%. Note that this gain only comes from a better usage of the demodulators.

Figure 5 shows the fairness performance against the number of nodes. From our definition above, the baseline MAX policy has a constant fairness of 1. As the number of nodes increases, the number of frames increases, and the probability to demodulate frames of large SFs drops significantly for all other policies. The fairness of the baseline FIFO policy drops to 0.75 for 1000 nodes. The fairness of FIFO-RR1 is slightly lower than that of FIFO: the difference reaches about 2% for 1000 nodes. This is because FIFO-RR1 demodulates more short frames than FIFO, as it demodulates short frames during long preambles. The fairness of FIFO-RR2 is significantly larger than that of FIFO: the gain exceeds 11% for 1000 nodes. This is because FIFO-RR2 plans the demodulation of large frames while it demodulates other frames.

FIFO-RR2 is thus able to improve both throughput and fairness compared to baseline FIFO. This means that the gain in throughput is not obtained by demodulating only short frames. Moreover, the gains improve as the number of nodes increases, showing the scalability of the proposal, while having limited additional computation and memory requirements.

C. Effects of the number of demodulators

Figure 6 shows the throughput performance against the number of demodulators, for 550 nodes. The number of demodulators has a significant impact on the performance of all policies (except for the MAX policy, which assumes an infinite number of demodulators). For baseline FIFO, the use of 4 demodulators instead of 8 decreases the throughput by 16%, while the use of 16 demodulators instead of 8 increases the throughput by 21%. The gap between baseline MAX and baseline FIFO for 8 demodulators exceeds 31%: this very large percentage shows the extent of throughput overestimation of research works that do not take into account the limited number of demodulators. The throughput of FIFO-RR1 always exceeds that of FIFO, while FIFO-RR2 always outperforms FIFO and FIFO-RR1. Compared to FIFO, FIFO-RR2 improves the throughput by 6.1% for 32 demodulators to 11.9% for one demodulator, with a percentage of 6.9%

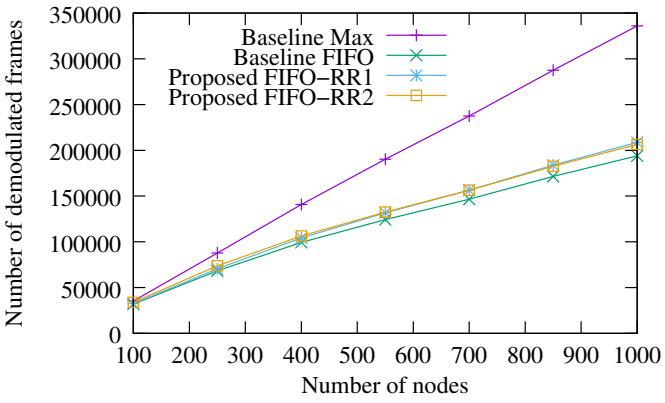

Fig. 4. Throughput performance as a function of the number of nodes.

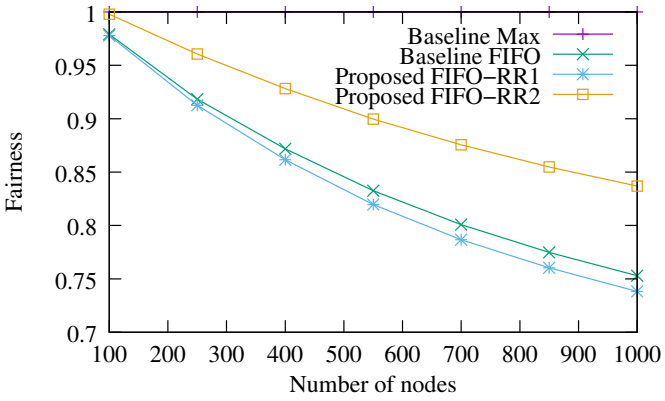

Fig. 5. Fairness performance as a function of the number of nodes.

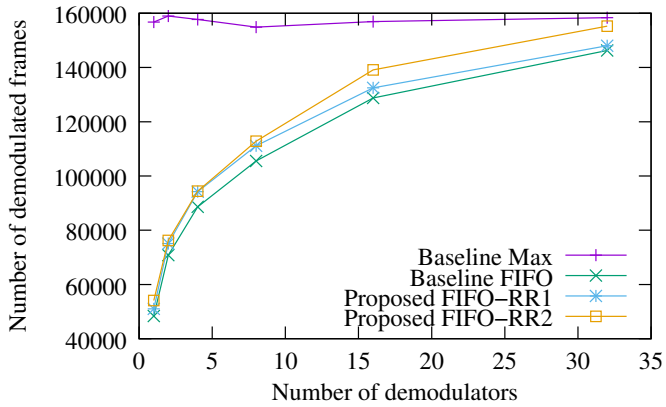

Fig. 6. Throughput performance as a function of the number of demodulators.

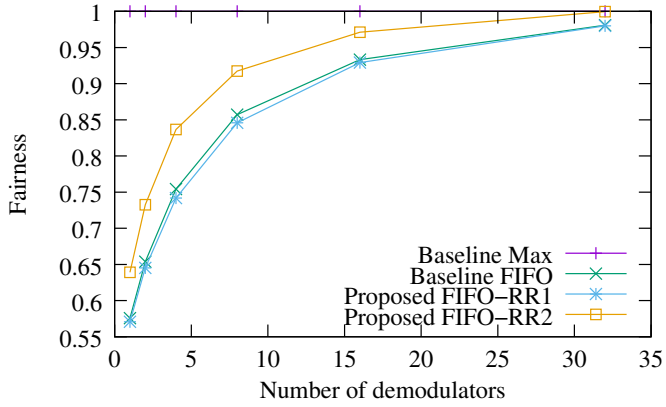

Fig. 7. Fairness performance as a function of the number of demodulators.

for 8 demodulators. This percentage reduces as the number of demodulators increases, as all policies attain maximum throughput with a sufficiently large number of demodulators.

Figure 7 shows the fairness performance against the number of demodulators. The fairness of FIFO-RR1 is slightly smaller than that of FIFO, while FIFO-RR2 significantly outperforms both FIFO and FIFO-RR1. The gain in fairness of FIFO-RR2 over FIFO is 7% for eight demodulators. Interestingly, the fairness level of FIFO-RR2 for x demodulators is close to the fairness level of FIFO for $2x$ demodulators, thereby achieving a two-fold saving of the number of required demodulators.

V. Conclusions and Perspectives

The hardware architecture of SX1301, used in most LoRa gateways, enables the demodulation of up to eight frames in parallel. As the traffic load of the network increases, this specific feature becomes a severe limitation. We propose two packet arbiter policies called FIFO-RR1 and FIFO-RR2, that aim to optimize the allocation of demodulators, based on the observation that demodulators are only busy when demodulating payloads, not during preambles. Thus, the demodulators can be recursively reused during preambles. FIFO-RR2 is able to demodulate more frames with large SFs than other policies. Simulation results showed that FIFO-RR2 jointly enhanced throughput and fairness even for a large numbers of devices, providing high scalability despite the stringent hardware constraint. In our future work, we will further enhance our proposed schemes with preemptive scheduling measures and conduct experimental evaluations.

References

[1] Semtech Corporation, "AN1200.22 LoRa Modulation Basics," Semtech, Application note Revision 2, 2015, accessed 2018-01-29. [Online]. Available: http://www.semtech.com/uploads/documents/an1200.22.pdf

[2] LoRa Alliance Technical Committee, "LoRaWAN 1.1 Specification," LoRa Alliance, Standard V1.1, 2017.

[3] Semtech Corporation, "Sx1301 datasheet - wireless & sensing products," Semtech Corporation, Datasheet, June 2017, v2.4.

[4] R. B. Sorensen, N. Razmi, J. J. Nielsen, and P. Popovski, "Analysis of LoRaWAN uplink with multiple demodulating paths and capture effect," in *IEEE International Conference on Communications (ICC)*, 2019.

[5] P. K. Dalela, S. Sachdev, and V. Tyagi, "LoRaWAN network capacity for practical network planning in India," in *URSI Asia-Pacific Radio Science Conference (AP-RASC)*, 2019.

[6] A. Rahmadhani and F. Kuipers, "When LoRaWAN frames collide," in *International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH)*, 2018, pp. 89–97.

[7] J. Haxhibeqiri, F. Van den Abeele, I. Moerman, and J. Hoebeke, "LoRa scalability: a simulation model based on interference measurements," *Sensors*, vol. 17, no. 6, p. 1193, 2017.

[8] A. Waret, M. Kaneko, A. Guitton, and N. El Rachkidy, "LoRa throughput analysis with imperfect spreading factor orthogonality," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 408–411, 2019.

[9] K. Mikhaylov, J. Petäjäjärvi, and T. Hänninen, "Analysis of capacity and scalability of the LoRa low power wide area network technology," in *European Wireless Conference*, 2016.