Server-Driven Scheduling for Scalable and Collision-Free LoRaWAN Networks

Participants
Christoffer Ejsing
Simon Juhl

Supervisors
Tatiana Kozlova Madsen
Rasmus Sibbern Frederiksen

Project period:
Spring Semester 2025

Page count:
42

AALBORG
UNIVERSITY

STUDENT REPORT

Computer Engineering
Aalborg University
June 4th 2025

Abstract

This project explores server-driven time-based scheduling protocols for LoRaWAN networks composed of devices that transmit periodically. The aim is to eliminate collisions and improve channel utilization by assigning time slots to devices according to a collision-free schedule. Three scheduling protocols are introduced, each with different approaches to slot assignment and rescheduling. A custom simulator is used to evaluate their performance under varying network densities and device periodicities.

Results show that all three protocols can maintain close to 100% utilization of available time slots under periodic traffic patterns, provided that drift correction and rescheduling are managed correctly. The project focuses on single-channel, single-spreading factor LoRaWAN scenario, but the proposed scheduling structure is designed to be extensible to more complex deployments. The findings highlight the importance of slot structuring, rescheduling strategies, and drift management in achieving reliable operation at scale in LoRaWAN systems.

Acronyms

CPA Compatability-Prioritized Assignment. 18, 20, 28, 30, 33, 35, 37, 40

CRC Cyclic Redundancy Check. 2, 8

GI Guard Interval. 14

IoT Internet of Things. 1, 4

LoRa Long Range. 12

LoRaWAN Long Range Wide Area Network. i, 1, 4, 5, 7, 9, 10, 12

MAC Medium Access Control. 1, 4, 7, 9

NSA Next Slot Assignment. 18, 30, 33, 35, 37, 40

SF Spreading Factor. 2, 4, 7

SIR Signal-to-Interference Ratio. 39

SWA Sliding Window Assignment. 20, 21, 28, 30, 33–37, 40, 41

ToA Time-on-Air. 2–4, 29

Contents

Chapter 1

Introduction

The rise of the Internet of Things (IoT) has transformed how data is collected and used. Across sectors thousands of small power constrained devices are being deployed to capture sensor data and communicate that data over long distances. As these systems grow in scale and importance, the demand for reliable and efficient communication technologies increases.

To support large-scale deployments across large geographical areas, it is essential to use wireless technologies that are capable of achieving long-range coverage. The devices that are a part of these systems are often battery powered such that they can be placed anywhere without having to connect the devices to a power source. They are designed to be able to operate for years on battery power, but depending on their tasks the energy use can vary. Therefore, energy consumption is a key constraint when choosing the wireless technology to base these networks on. Long Range Wide Area Network (LoRaWAN) is one of several Low Power Wide Area Network standards which is designed to achieve a low power consumption, long transmission range, while being able support thousands of devices per gateway.

However, while LoRaWAN enables long-range communication with little energy use, its network architecture presents significant challenges as device density increases. The default Medium Access Control (MAC) protocol in LoRaWAN relies on an ALOHA-based access scheme, where devices transmit without sensing or coordinating access to the channel. This approach works in sparse networks but in more dense environments the extra communication increases the risk of packet collisions, and when the collision rate goes up the availability of data decreases. In a future where more and more IoT devices share the same spectrum the density will go up and the problem will get worse. Therefore, a solution to this problem is required.

Chapter 2

Technical analysis

2.1 LoRa

LoRa is a proprietary modulation technique based on chirp spread spectrum modulation. It encodes information in signals that linearly increase or decrease in frequency over time. These are referred to as chirps or symbols. Each chirp sweeps across a range of frequencies in a set duration, called the symbol time, which is determined by the Spreading Factor (SF). LoRa defines six spreading factors, SF7 to SF12, which directly influence the transmission properties.

Lower spreading factors result in faster chirps, leading to shorter symbol durations and higher data rates. However, they are less robust to noise and allow communication only over shorter distances. In contrast, higher spreading factors increase the symbol duration, slowing down the transmission rate but significantly extending the communication range. This trade-off makes the choice of SF a crucial design consideration in LoRa networks.

2.1.1 Packet Format and Transmission

LoRa supports two physical layer packet formats: implicit and explicit mode. In implicit mode, the receiver must already know the packet configuration: payload size, coding rate, and Cyclic Redundancy Check (CRC) presence. Explicit mode on the other hand includes this information in the packet header, making it better suited for networks with different device configurations.

The format for explicit mode is shown in figure 2.1. It begins with a preamble to help the receiver synchronize with the incoming transmission, followed by a physical header and its CRC, and finally the main payload and a final CRC.[1]

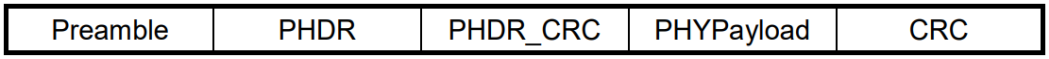


Figure 2.1: LoRa Physical Layer Packet Format

2.1.2 Time-on-Air and Transmission Duration

The time required to transmit a LoRa packet, referred to as Time-on-Air (ToA), depends primarily on three parameters: the spreading factor, the bandwidth, and the coding rate. The SF controls the duration of each symbol, with higher values resulting in longer symbol times and thus greater ToA. The bandwidth influences how quickly information can be transmitted, and the coding rate adds redundancy for forward error correction.[2]

The coding rate in LoRa is defined as $CR = 4/(4+n)$, where $n \in \{1, 2, 3, 4\}$. It determines how many bits are used for error correction in each block of 4 data bits. For instance, CR = 4/5 means that for every 4 bits of data, 1 bit of error-correcting code is added. This makes the transmission more robust to noise and interference but also increases the total number of bits that must be transmitted, which in turn increases the ToA.[3]

The following formula from the is used to compute the ToA[4]:

$$T_{\text{packet}} = T_{\text{preamble}} + T_{\text{payload}} \tag{2.1}$$

where

- $T_{\text{symbol}} = \frac{2^{SF}}{BW}$
- $T_{\text{preamble}} = (n_{\text{preamble}} + 4.25) \cdot T_{\text{symbol}}$
- $T_{\text{payload}} = N_{\text{payload}} \cdot T_{\text{symbol}}$

The payload symbol count N_{payload} is computed as:

$$N_{\text{payload}} = 8 + \max\left(\left\lceil \frac{8 \cdot PL - 4 \cdot SF + 28 + 16 - 20 \cdot H}{4(SF - 2 \cdot DE)} \right\rceil \cdot (CR + 4), 0\right) \tag{2.2}$$

where:

- PL is the payload size in bytes (12 in this case),
- $H = 0$ for explicit header mode,
- $DE = 1$ if low data rate optimization is enabled (usually when SF $\geq$ 11),
- CR is the coding rate denominator offset, i.e., for CR = 4/5, use $CR = 1$.

Figure 2.2 illustrates the resulting ToA for a 12-byte payload transmitted using different spreading factors, assuming a 125 kHz bandwidth and CR = 4/5.

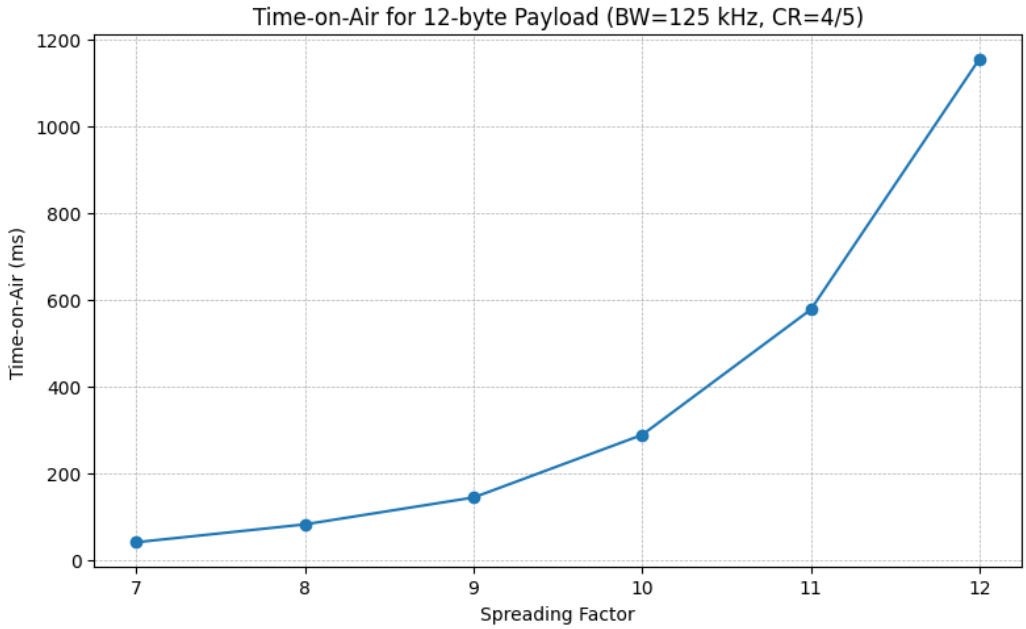


Figure 2.2: Time-on-Air of a 12-byte LoRa packet using 125 kHz bandwidth and CR=4/5 across different spreading factors

2.1.3 Receiver Sensitivity and Robustness

One of LoRa's strengths lies in its ability to operate reliably at very low signal levels. As the SF increases, the receiver sensitivity improves, enabling communication over longer distances and through more obstacles. The table below shows the approximate receiver sensitivity levels across spreading factors when using a 125 kHz bandwidth[2].

Spreading Factor	Receiver Sensitivity (dBm)
SF7	-123
SF8	-126
SF9	-129
SF10	-132
SF11	-134.5
SF12	-137

Table 2.1: Approximate receiver sensitivity by SF (125 kHz bandwidth)

These values also reflect LoRa's robustness against interference. Higher spreading factors provide stronger processing gain, making it easier for the receiver to recover data when portions of the signal are distorted or lost. The combined influence of ToA, sensitivity, and robustness underpins many design trade-offs. Higher SFs may reduce network capacity due to increased airtime but enable longer communication ranges and stronger link reliability.

2.2 LoRaWAN

LoRaWAN is the MAC layer protocol that operates on top of the LoRa physical layer and defines the network architecture and communication behavior of LoRa-based systems. It is widely used in IoT due to the long communication range and interference resistance of LoRa modulation. LoRaWAN organizes the network into a star-of-stars topology, where multiple end devices communicate with one or more gateways, which then relay the data to a central network server. The server, in turn, forwards the data to an application server where it is processed.

The main components of a LoRaWAN network include:

- End Devices: Sensor nodes that transmit data periodically or on demand.
- Gateways: Relay devices that forward LoRa-modulated messages to and from the network server using IP-based backhaul communication.
- Network Server: Responsible for packet deduplication, security validation, and coordination of downlinks.
- Application Server: Receives validated data for application-specific processing.

An overview of the LoRaWAN architecture is illustrated in figure 2.3.

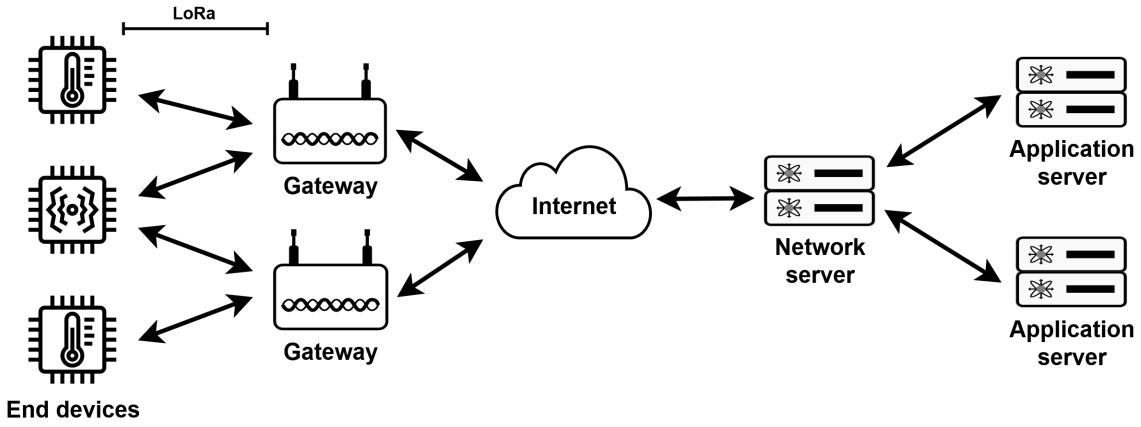


Figure 2.3: LoRaWAN network

2.2.1 Communication Behavior and Device Classes

Communication in LoRaWAN is initiated by end devices through uplink transmissions that carry sensor readings or status updates. After each uplink, the end device opens one or two receive windows to potentially capture a downlink. Downlinks are used for acknowledgements or server-issued commands, but can also carry information in the PHYPayload which can be used for device-specific functionality which builds on top of the LoRaWAN protocol.

LoRaWAN defines three classes of end devices which each have different energy and synchronization characteristics:

- Class A devices operate in the most energy-efficient mode. They enter sleep mode when not actively transmitting or receiving. After each uplink transmission, they open two receive windows (RX1 and RX2) where they briefly listen for a downlink preamble. Downlink communication is only possible in these windows, meaning the network must delay responses accordingly. This class is optimal for battery-powered devices with low-latency requirements.
- Class B adds scheduled receive windows by synchronizing to network beacons. This allows the server to initiate downlink transmissions at specific times in contrast to the basic Class A behavior. However, since this requires the device to stay in receive mode more frequently than Class A it is also more energy consuming than Class A.
- Class C devices are continuously listening unless they are transmitting. This results in the lowest downlink latency, but it comes at a high energy cost and is therefore not very suitable for devices which are battery powered.

The tradeoff between downlink latency and energy consumption is a critical factor when selecting a LoRaWAN device class. Class A offers the lowest energy consumption by design, as devices spend most of their time in sleep mode and only briefly wake up to transmit data and listen for downlink messages. In contrast, Class B and especially Class C increase responsiveness at the cost of significantly higher energy usage.

The impact of these class-specific behaviors can be better understood by examining the power consumption characteristics of the Semtech SX127x transceiver, as shown in Table 2.2. The power draw varies substantially depending on the device's current operation mode.

Operation mode	Power consumption
Sleep mode	0.2 μA
Idle mode	1.5 mA
Standby mode	1.6 mA
Receive mode (high power)	11.5 mA
Transmit mode (low power)	120 mA

Table 2.2: Power consumption data for different modes of operation for the LoRa SX127x transceiver

As the table highlights, transitioning from sleep mode to receive mode increases current draw by a factor of over 57,000. This makes the energy cost of continuous or periodic listening, as required by Class B and Class C, substantially higher than for Class A devices. In Class A, energy consumption remains low because the transceiver operates in high-power modes only briefly during the uplink transmission and the short receive windows. This behavior is illustrated in figure 2.4, where the timing and duration of downlink opportunities are clearly constrained.

By minimizing the time spent in high-power modes, Class A ensures long battery life, making it ideal for applications where infrequent communication and energy autonomy are prioritized.

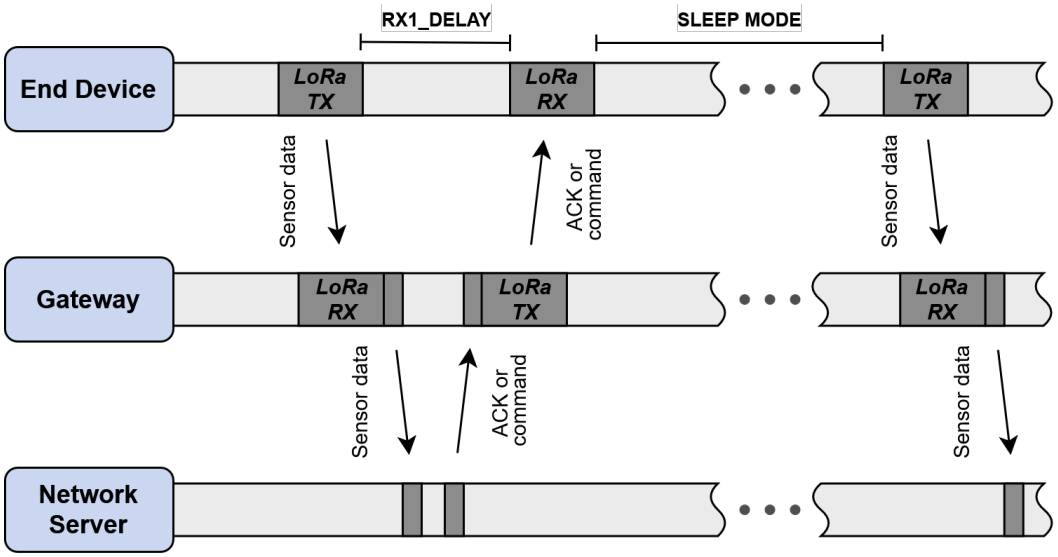


Figure 2.4: Communication between device and network server initiated by the device (Class A)

2.2.2 Protocol Constraints and Extension

LoRaWAN's MAC layer strictly defines the structure of uplink and downlink opportunities. Once the downlink message is received by the end device, if the transmission is not a LoRaWAN acknowledgment, it may still carry application-specific instructions. These are not interpreted as part of the LoRaWAN protocol itself but must be handled by custom logic implemented at the device's application layer.

Since the protocol cannot be directly modified, extending LoRaWAN must occur by layering additional logic on top of the existing communication model. This allows the application layer of the end device to interpret and react to the content of downlink messages.

2.2.3 Frequency Band and Payload Structure

In Europe, LoRa operates in the unlicensed 868 MHz ISM band, which ranges from 863 MHz to 870 MHz. This band provides nine channels for chirp spread spectrum transmissions, eight for uplinks, and all nine for downlinks. This flexibility supports high-density deployments.

2.3 Clock Drift and Timing Uncertainty

Clock drift in LoRaWAN devices arises from the frequency inaccuracies of their oscillators. These oscillators can deviate from their expected frequency due to factors like temperature variations, aging, and other environmental conditions. Such deviations lead to a difference between the device's internal clock and real time.

Over time, the microcontroller's drift can accumulate, resulting in increasing misalignment with real time. In some systems, the device's perception of time is not important, but in others it is crucial for devices to transmit at specific times. For instance, if device transmissions must fit into a schedule, then excessive drift can cause a device to transmit too early or too late relative to others. This misalignment can lead to overlapping transmissions and packet collisions, resulting in data loss.

One microcontroller that could be used in such a system is the ESP32, which according to its documentation has a clock drift of ± 10 ppm [5]. This means the clock can drift up to 10 microseconds per second, accumulating to approximately 0.864 seconds per day.

To mitigate the effects of clock drift, synchronization mechanisms are often used. These include, for example, periodic synchronization beacons, as seen in LoRaWAN Class B devices. However, Class A devices can also receive clock drift correction information through downlink messages.

2.4 LoRa Collisions

Due to the limitations imposed by the laws of nature with regard to time and frequencies, there is a natural upper limit to the number of LoRa transmissions that can occur simultaneously. These restrictions are unavoidable with current technology. Therefore, it is only natural that one would want to utilise the medium efficiently, in order to achieve the highest possible throughput for the network. Now, as stated previously, LoRa has certain properties that allow for better utilization of the medium, by adding orthogonal spreading factors that ensure robustness in relation to interference, which gives the opportunity to have multiple transmissions happen simultaneously.

However, LoRaWAN utilizes a Pure ALOHA MAC protocol. This means that transmissions can happen any time for a given end device, which of course leads to collisions, in cases where they share spreading factor and channel. This effectively lowers the overall throughput of the network, and by utilizing Pure ALOHA, LoRaWAN allows for uncoordinated transmissions at all times leaving the network inefficient. These problems however, may not arise from networks with a couple of hundred sensors, but when the system is scaled such that thousands of sensors are deployed over a relatively small geographical area the problem of avoiding collisions increases.

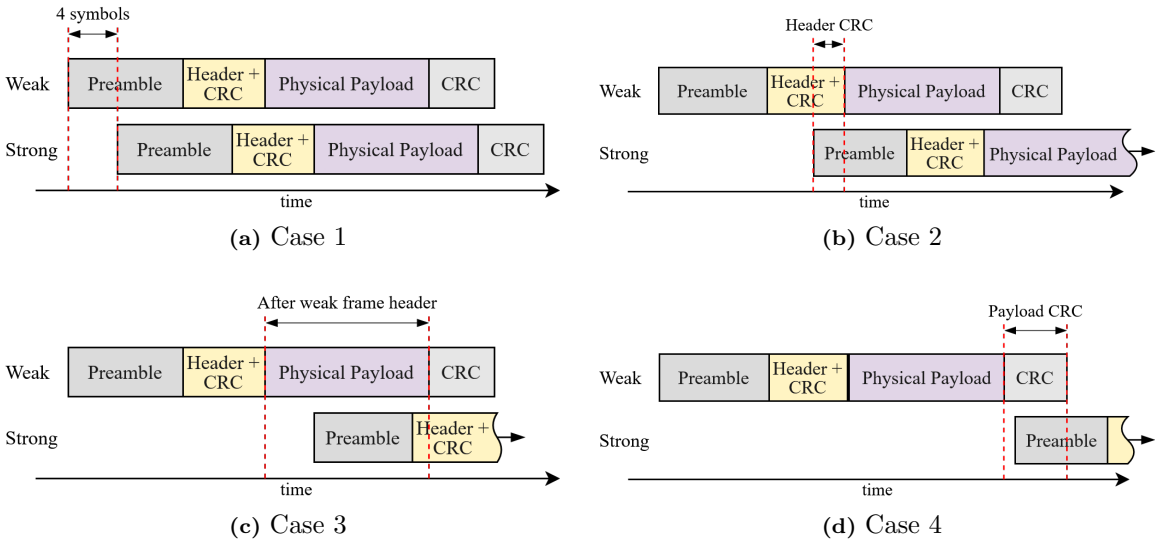
While transmissions on different SFs have a significantly lower probability of collisions, the probability is not zero. The SFs are not completely orthogonal and therefore some inter-SF interference still exists. According to [6] the impact of the interference depends on which SFs the interfering signals are using and the signal-to-interference ratio between the two signals. They determine minimum SIR thresholds for when the probability of correct demodulation

starts to degrade. They find that for a SX1272 transceiver which received a reference signal with SF12 and an interference signal with SF7 the SIR threshold is -25 dB, but when using a reference signal with SF7 and an interference signal with SF12 the SIR threshold is -9 dB. Thus, lower SFs are more vulnerable to co-channel inter-SF interference.

The results in [7] show that when spreading factors are distributed across all end devices in a optimal manner, network throughput greatly increases compared to when spreading factors are randomly assigned. The assignment strategy also greatly impacts the scalability of the system.

Overlapping frames can result in different outcomes depending on the timing and signal strength of the colliding transmissions. One option is that both packets are lost but there is also the option that the receiver successfully demodulates one of the packets. This is possible due to the capture effect which allows the receiver to reject the weaker signal while receiving the stronger signal[8]. The receiver's ability to correctly do this is dependent on the difference in the RSSI. According to Semtech this difference in RSSI needs to be more than 6 dB, otherwise it is not certain that the receiver can distinguish the signals from each other[4]. If the stronger signal starts before or at the same time as the weak signal, then the reception of the strong signal will continue without interference. But if the weak signal starts first, then the offset of the strong signal determines what the outcome of the collision is. According to [9] there are four different cases:

1. Case 1: Both frames are lost if the stronger signal arrives after the receiver locks onto the weak signal during preamble. The lock is obtained after 4 preamble symbols.
2. Case 2: The stronger signal is received if it arrives during the header CRC of the weaker signal. It will interfere with the reception of the CRC, the weaker signal's header will seen as having errors and the lock on the weaker signal is released.
3. Case 3: Both frames are lost if the stronger signal arrives after the weak signal's header is received. The receiver stays locked onto the weak signal and the strong signal corrupts the payload.
4. Case 4: The weaker signal can sometimes still be received if the stronger signal only overlaps slightly with the payload CRC. This is because of the error correction techniques of the LoRa receivers.



(a) Case 1 **(b)** Case 2

(c) Case 3 **(d)** Case 4

Figure 2.5: Capture effect cases. Each case shows the weak packet on top and the strong packet on the bottom

2.5 Collision Avoidance

A lot of research has been conducted into how to improve the performance of a LoRa network by reducing the number of collisions. This research has focused on different approaches, which can be categorized as follows:

- Spreading factor and channel allocation
- Carrier-sensing MAC protocols
- Scheduled transmission coordination

Each category offers a distinct approach to deal with contention: spreading transmissions apart across orthogonal resources, avoiding conflicts through sensing, or directly coordinating the timing of each device's transmissions. Below, these different approaches will be explored in order to get an understanding of their advantages and disadvantages.

2.5.1 Spreading Factor and Channel Allocation

One of the simplest ways to reduce collisions in LoRaWAN is to distribute the devices more effectively across the available spreading factors and channels. This approach leverages the fact that transmissions with different spreading factors are quasi-orthogonal, meaning they can often be received simultaneously without interference. Likewise, using different frequency channels can prevent overlapping transmissions from colliding.

The assignment of SFs and channels can be performed by randomly assigning devices to different SFs or by assigning the devices based on their distance to the gateway to compensate for the signal attenuation. However, smarter allocation schemes can have a significant effect on performance. For example, the interference-aware assignment scheme presented in [7] dynamically monitors the network and reassigns devices to SFs that experience less congestion. If a device transmits using an SF that frequently suffers interference, the gateway can reassign it to a less utilized SF.

This approach was tested against two baselines: a random assignment strategy, and a distance-based one where devices farther from the gateway are assigned more robust spreading factors. The results showed clear advantages to intelligent assignment. In a single-channel scenario with 4000 devices, the random scheme resulted in less than 10% packet success, while the interference-aware scheme achieved over 55%. With only 500 devices, the improvement was even more dramatic: from 50% success with random assignment to over 90% with interference-aware assignment.

Adding more channels further boosted performance. Moving from one to three channels with interference-aware SF assignment improved success rates by up to 20 percentage points in dense scenarios.

These results confirm that SF and channel diversity are powerful tools for improving LoRaWAN performance. However, this method alone cannot eliminate collisions. If two devices with the same SF transmit on the same channel at the same time, they will still interfere with each other. This is especially problematic in dense networks, where the number of possible orthogonal configurations will get exhausted. Further coordination is needed to avoid time-domain conflicts.

2.5.2 Carrier-Sensing MAC Protocols (CSMA)

To address the problem of overlapping transmissions directly, some approaches modify the MAC behavior of end devices as an alternative to Pure ALOHA which LoRaWAN uses. Carrier Sense Multiple Access introduces a smarter alternative: Before transmitting, a device

first listens to the channel to determine if it is idle. If the channel is free, the device proceeds with transmission. If the channel is busy, the device either tries to listen again immediately or backs off for a random period.

In [10], a non-persistent CSMA protocol is proposed for LoRaWAN. Each device is assigned a subset of available channels. Before transmitting, it checks whether any of its assigned channels are free. If all are busy, the device waits for a random backoff interval before trying again. This design allows for more efficient use of the channel resources and reduces the chance of immediate collisions.

To evaluate the effectiveness of this approach, the study compared CSMA with a random frequency-hopping scheme, just like ALOHA but across multiple channels. Devices attempted to transmit large payloads every 30 minutes using SF12, and the key performance metric was the Transmit-to-Target Ratio (TTR), which is a ratio describing the number of transmissions needed in order to transmit a set number of packets. If the ratio is close to 1, it symbolizes that the protocol in question is efficient and reliable in delivering packets error free.

With 200 devices, the random hopping method required 50% more transmissions than CSMA. At 1000 devices, the difference was even bigger: CSMA maintained a TTR below 1.5, while the baseline exceeded 3. This shows that CSMA greatly reduces retransmissions and improves network efficiency.

However, CSMA is still reactive rather than preventive. It works well when contention is low or moderate, but cannot handle perfectly simultaneous transmissions, which may still occur frequently in large-scale systems. Since LoRa devices often wake up and transmit at intervals, many devices can accidentally align their schedules, triggering collisions that CSMA cannot prevent. Thus, more structured coordination is needed.

2.5.3 Scheduled Transmission Coordination

A more proactive way to prevent collisions is to coordinate the transmission schedule of end devices. This strategy works by assigning each device to a specific time slot during which it is allowed to transmit. If all devices follow their assigned schedule, collisions can be eliminated altogether.

A system that work this way is described in [11], which proposes a scheduling and synchronization scheme tailored to LoRaWAN. End devices periodically send synchronization requests to the gateway, indicating their desired uplink interval and clock accuracy. The server then calculates and assigns time slots for both data transmissions and future resynchronization. It also sends back the clock offset needed to align the device's schedule.

This architecture enables tight control over channel access. To further reduce collisions, out-of-band synchronization is used in some configurations, ensuring that control messages do not interfere with data traffic. The results show substantial improvements: in some cases, the packet delivery ratio increased by 30% compared to a baseline using ALOHA.

Despite its advantages, this approach introduces new costs. Devices must send control messages to maintain synchronization. Clock alignment requires regular updates, especially in the presence of drift.

2.6 Problem Statement

This project investigates how to reduce packet collisions in LoRaWAN networks while avoiding the overhead associated with fully coordinated scheduling systems. LoRaWAN's default

Pure ALOHA channel access results in high collision rates as the number of devices grows, particularly when using a single channel and spreading factor. Multi-channel and multi-SF solutions have been investigated extensively and been proven very beneficial to the performance of a network. Therefore, this aspect of utilizing multiple channels and spreading factors will not be considered in this project, which is why the focus will be on how to reduce the number of collisions on a single channel and single spreading factor network.

Many of the most effective solutions rely on frequent time synchronization and control messages to explicitly reserve transmission slots. These coordinated schemes result in high communication overhead. This motivates a search for alternative approaches that provide meaningful coordination with lower cost.

This project therefore explores a scheduling approach for LoRaWAN where:

- Devices transmit periodically with potentially different periods
- All coordination is handled by the network server using downlinks only after successful uplinks in Class A
- Collisions are avoided by finding stable schedules and rescheduling
- The system scales to high densities while remaining energy-efficient and minimally disruptive

The goal is to evaluate whether such a model can maintain high reliability and throughput while minimizing coordination overhead.

Problem statement: How can a LoRaWAN network reduce collisions through server-driven scheduling, while remaining scalable, and robust to variation in device periodicity and drift?

Chapter 3

Protocol Design

As initially described in section 2.2, a LoRaWAN network consists of end devices that gather some form of information like sensor readings, which they communicate back to an application server that aggregates the information of all the end devices that a specific owner has. The communication from end devices to application server goes through gateways that act as intermediaries between LoRa-based communication and IP-networks. Gateways forward the packets to a network server, which ensures that there are no duplicated, forged or otherwise altered packets before they arrive at the application server. An overview of a typical LoRaWAN network can be seen in figure 3.1.

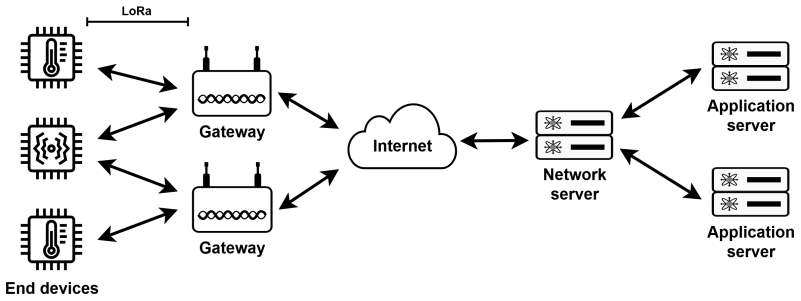


Figure 3.1: Classical depiction of a LoRaWAN network

From the depiction on figure 3.1, it should be clear that the central entity in the system is the network server, it is aware of all devices that have joined the network and ensures that rouge devices cannot introduce false information into the application server. This makes it an obvious choice for a central scheduling entity.

When a new device joins the network, it exchanges messages with the join server and derives session keys, which are used for the remainder of the communication between the end device and the network server. When a device has joined the network, it can transmit packets whenever using the ALOHA protocol.

3.1 Slot-Based Scheduling and Period Alignment

The protocol is designed for use in LoRaWAN networks, where multiple devices transmit periodically to the application server. Due to the periodicity of the devices, it is possible to shift transmissions such that the number of collisions is limited or even eliminated, depending on the number of nodes in the system and their periodicity.

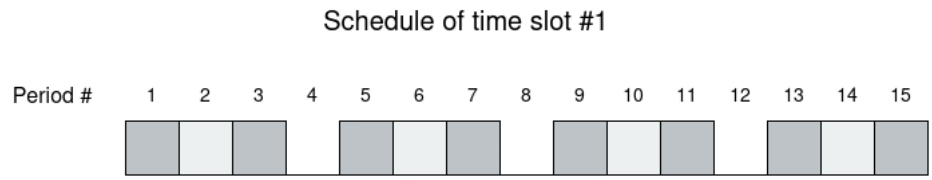


Figure 3.3: An example of devices sharing the same time slot with compatible periods

This section elaborates on how these shifts occur and how an optimal schedule is achieved by exploring the structure of time slots, the scheduling of devices, and the network server's strategy for handling drift.

3.1.1 Time Slot Structure and Period Compatibility

To ensure collision-free transmissions, devices are assigned a time slot in which only that specific device is transmitting. The schedule can therefore be seen as an infinite series of time slots starting from time 0. However, to further structure the scheduling of the protocol, a minimum period is defined, where a minimum period contains a certain number of time slots.

This means that instead of seeing the schedule as an infinite series of time slots, it can now be viewed as a finite number of time slots that repeat after a given time. An illustration of this can be seen in Figure 3.2.

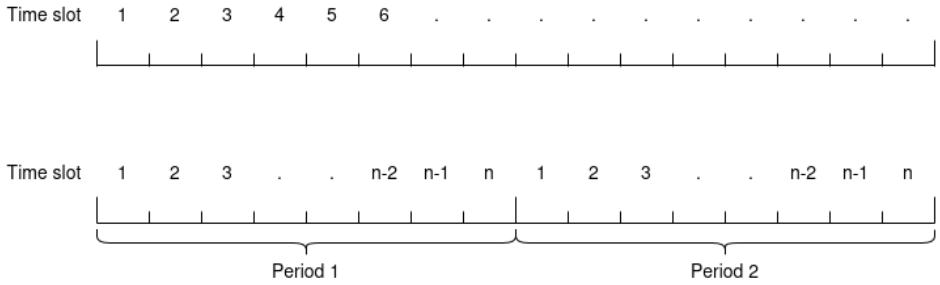


Figure 3.2: A schedule of an infinite number of time slots and a schedule consisting of periods with n time slots

A minimum period specifies the minimum supported device transmission periodicity, and only periods that are a multiple of the minimum period are supported. This constraint is enforced to ensure that each scheduled device appears in the same time slot for each transmission.

This structure benefits the efficient use of time slots. Since all end devices in the system have a period that is a multiple of a minimum period, they will always transmit in their originally assigned time slot. Additionally, devices that do not transmit every minimum period can share a time slot with other devices if their periods are compatible. This is illustrated in Figure 3.3, where a device with a period of 2 and a device with a period of 4 both share the same time slot.

3.1.2 Slot Duration and Guard Interval

To secure collision-free transmissions, a time slot must be as long as the transmission. As mentioned in Section 2.2, there are both uplink and downlink communications in LoRaWAN. In this project, both are included in a time slot so that no additional scheduling is needed for downlink communication between the network server and the devices.

The transmission time includes time for the uplink, the delay before the device enters listening mode, and time to receive the downlink. A Guard Interval (GI) is added to make the system more robust to drift.

$$\mathrm{T_{slot}} = \text{uplink} + \text{rx delay} + \text{downlink} \tag{3.1}$$

Illustrative values:

- Uplink transmission time: 1.5 seconds
- Downlink transmission time: 1.5 seconds
- Receive delay: 1 second

$$\mathrm{T_{slot}} = 1.5 + 1.5 + 1 = 4 \text{ seconds} \tag{3.2}$$

To calculate a suitable guard interval, we consider how much devices can drift over time and how long the system should run without requiring a drift correction. For example:

- Rescheduling bound: 12 hours $= 43,200,000,000\ \mu s$
- Drift: 10 ppm $= \frac{10}{1{,}000{,}000}$

$$\text{Minimum GI} = 2 \cdot \lfloor \text{drift} \cdot \text{rescheduling bound} \rfloor \tag{3.3}$$

$$\text{Minimum GI} = 2 \cdot \left\lfloor \frac{10}{1,000,000} \cdot 43,200,000,000 \right\rfloor = 864,000 \tag{3.4}$$

Then the number of slots per period is:

$$sc = \left\lfloor \frac{T_{min}}{\mathrm{T_{slot}} + \text{Minimum GI}} \right\rfloor \tag{3.5}$$

Where T_{min} is the defined minimum period of the system.

$$\begin{aligned} T_{min} &= 1,000,000 \cdot 60 \cdot 5 \\ \mathrm{sc} &= 61 \\ \text{slotduration} &= 4,000,000\ \mu s \\ GI &= \frac{T_{min} - (sc \cdot \mathrm{T_{slot}})}{sc} = 918,032.79\ \mu s \end{aligned}$$

The choice of rescheduling bound directly influences the required size of the guard interval. A longer bound allows devices to operate longer without the server having to take any actions, but this increased drift tolerance comes at a cost: the guard interval must be longer. A longer guard interval reduces the number of usable time slots in a fixed-length period, thereby limiting how many devices can be scheduled. Figure 3.4 illustrates this trade-off between guard interval size and scheduling capacity for different rescheduling bounds.

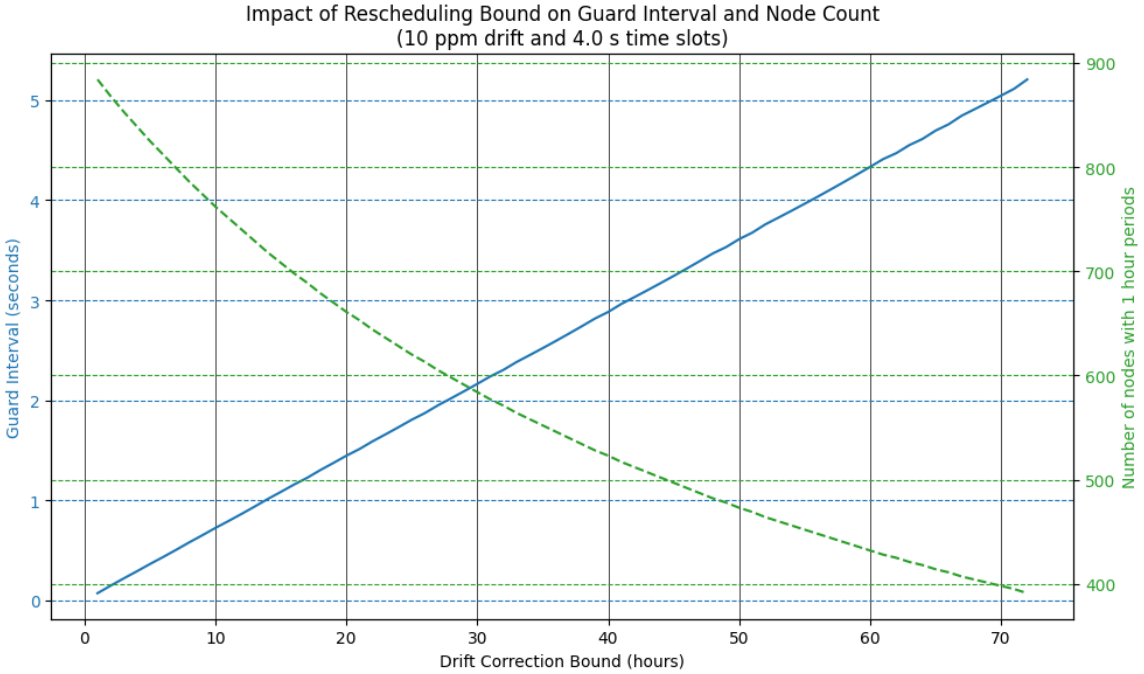


Figure 3.4: Effect of rescheduling bound on guard interval and supported number of nodes with hourly periods. As the rescheduling bound increases, the guard interval must grow to accommodate drift. This reduces the number of time slots available in a fixed-length period, thus limiting the number of nodes that can be scheduled without collisions.

3.1.3 Drift Tolerance and Passive Schedule Tracking

Even with guard intervals, drift may cause a device's transmission to slip into the next time slot. To prevent this, the network server must occasionally correct the drift.

The proposed system does not require additional communication from the device to measure drift. Instead, it takes advantage of the predictable periodicity of devices. Because the network server knows each device's period and its assigned time slot, it can estimate the expected arrival time of the next transmission.

The location of each device is also assumed to be known, allowing the server to account for propagation delay. The network server is able to compare the actual arrival time of a packet to its expected arrival, thereby calculating any deviation, i.e. drift. This process does not require explicit synchronization or polling from the device. This strategy reduces control overhead and energy usage while still supporting drift correction.

This passive tracking model makes it possible to support a large number of devices while minimizing communication overhead, especially compared to systems that rely on explicit resynchronization messages.

3.2 Random Assignment Version

This version of the protocol uses a simple scheduling approach where devices are assigned to time slots using a random selection strategy. While the selection strategy is not optimized for collision avoidance, the protocol is still capable of correcting drift and performing rescheduling. The simple scheduling approach also makes this version well-suited for comparison with other version which have the ability to coordinate and optimize the scheduling, in order to see how this impacts the performance of the network.

When a new device joins the network, it includes its requested transmission period in the uplink message. The network server then randomly selects a time slot from a time interval and assigns the device to that slot. For the device to align its transmission time with the time slot a time shift is calculated using the time shift calculations in section 3.1 and send to the end device.

Each time a device is assigned to a slot the network server evaluates all devices in that slot to determine whether any future collisions will occur in the schedule. As explained in the section 3.1 the device periods are multiples of the minimum period allowing them only to collide with devices within the same slot. Thus, the calculation of future collisions can be treated as a slot-wise problem.

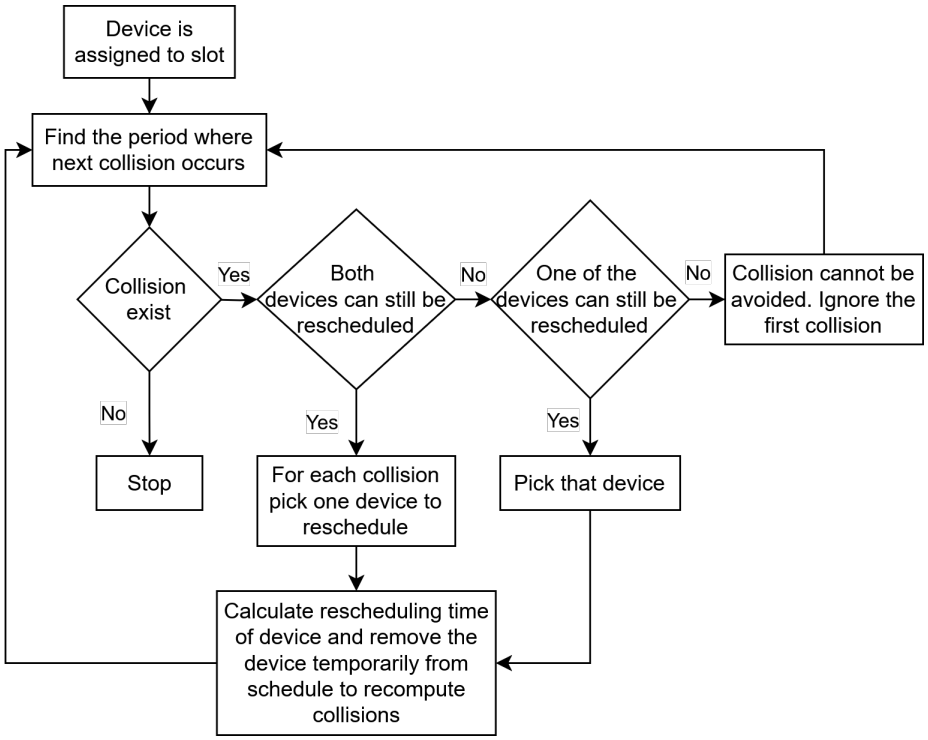


Figure 3.5: Finding collisions and marking devices for rescheduling

The code aims to obtain a collision-free time slot by marking devices for rescheduling before they cause a collision. Every time a device is marked for rescheduling it is temporarily removed from the schedule while the code checks for collisions among the remaining devices. When no more collisions can be found the network server does not need to do more before next slot assignment.

The network server keeps track of when all devices have to be rescheduled and drift corrected, handles the necessary tasks and sends the downlink message with the time shift. In figure 3.6 the flowchart of the entire protocol can be seen.

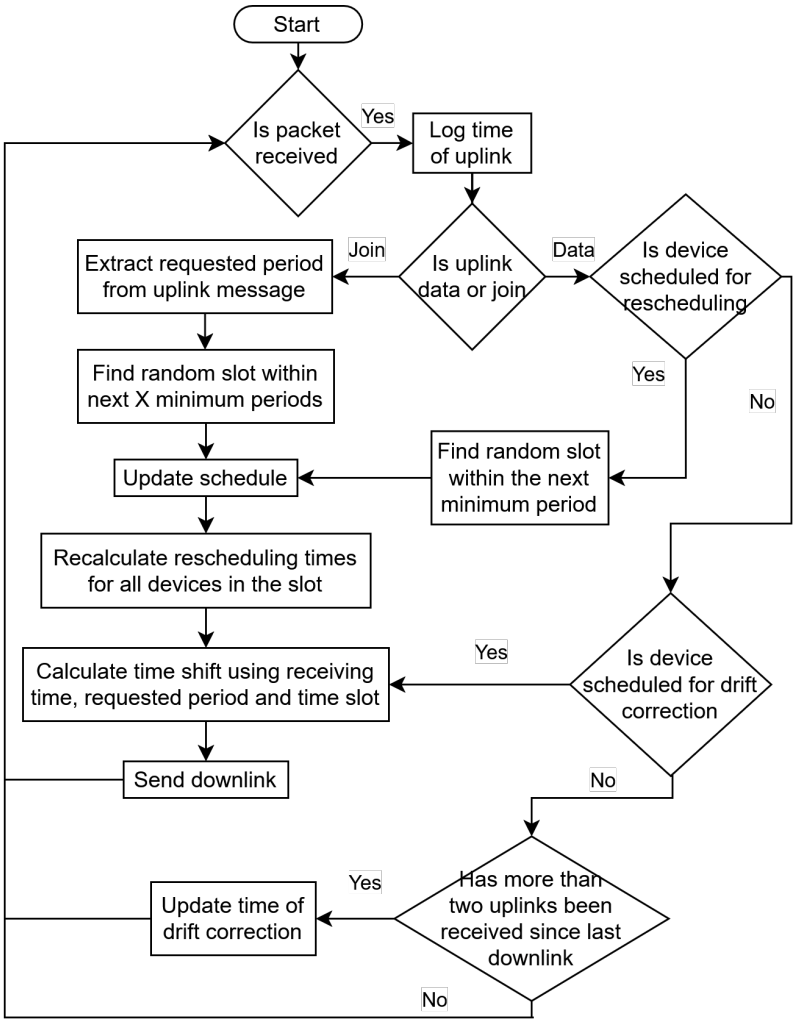


Figure 3.6: Flowchart of random assignment version

The primary disadvantage with this protocol is that it relies entirely on successful uplink receptions for it to handle rescheduling and drift correction. If a collision occurs due to random assignment, the network server will not receive any of the uplink messages making the server unable to send the downlink update. The next version will look at solving this.

3.3 Next Slot Assignment (NSA)

Building on the initial random assignment approach, this version of the protocol introduces a more deliberate strategy where devices are assigned to the next available slot in the schedule. The primary motivation for this change is to improve the reliability and maintainability of the schedule by minimizing the risk of collisions. In contrast to the earlier version, which tolerated packet loss and limited control over the network, this protocol aims to proactively avoid collisions. This will help to avoid the scenario where a single missed uplink prevents the server from sending its critical rescheduling and drift correction messages.

The version works similarly to the random assignment version with the exception that when a new device joins the network or needs to be rescheduled, the server searches for the next available time slot using the procedure seen in figure 3.7. The procedure begins by finding the next time slot in the schedule and then find that same time slot multiple minimum periods into the future given the device's periodicity. This period will be assigned to the `check_offset` value which is the number of minimum periods from the current period which

network server will then use as a starting point for the search for a free slot. After finding the time slot to start with the procedure incrementally searches forward in time for the next period where that slot is unoccupied. For each time slot, the server checks whether any of the devices already assigned to that slot will be transmitting at that future point in time. To calculate whether a device which is already assigned to a slot is going to transmit at the `check_offset` the difference between the `check_offset` and the offset of the device is found and if the difference modulo the device period, then the device is using the time slot at that period:

```
(check_offset-offset_assigned_dev) % period_assigned_dev == 0
```

If the statement is true, then the search moves to the next time slot or the next period until an available time slot is found.

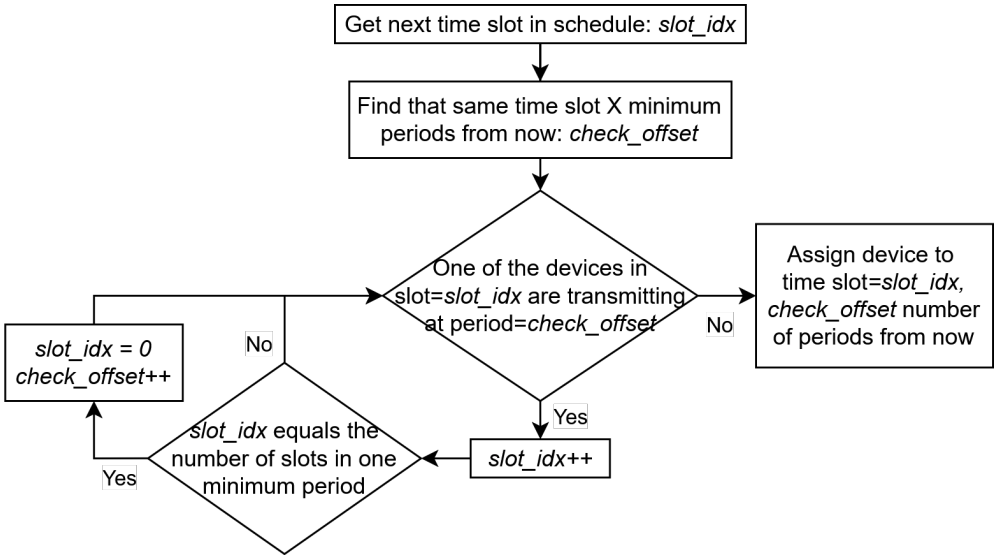


Figure 3.7: Flowchart of the NSA

This method increases the predictability of the network behavior. It allows the server to maintain a global schedule and ensures that downlink messages can be reliably delivered. The cost of this added reliability is a potential increase devices deviating from their desired transmission periodicity at every rescheduling, as the server may need to look ahead several minimum periods before finding an available slot. This deviation may increase for networks with more devices, since the network achieve higher utilization which makes it harder to find free slots.

Also, since this assignment strategy only checks whether the next slot is collision-free the device might not be long-term compatible with the other device periods in the slot resulting in more rescheduling. The next section will delve into the possibility of achieving more compatible schedules.

3.4 Compatibility-Prioritized Assignment (CPA)

In CPA the aim is to assign devices to time slots such that they are compatible with each other. Since devices will always use the same time slot unless they are rescheduled, the devices just have to be compatible within the same time slot. Therefore, there will often be a time slot where a device's period will be compatible with the other devices in the slot, at least for lower density networks. However, at some point when networks get more dense it is not possible to find compatible slots anymore. In that case the protocol will try to find the

time slot assignment with the longest lifetime. The assignment procedure is shown in figure 3.8.

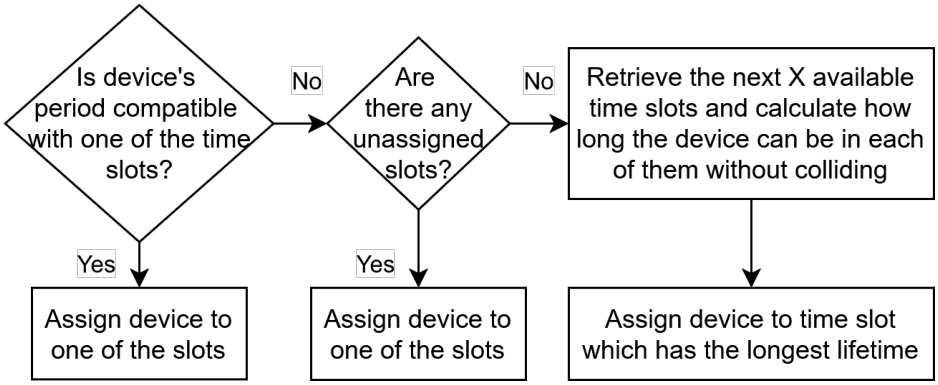


Figure 3.8: Compatibility assignment procedure

In order to find out whether the new device is compatible with all the periods in the time slot it is necessary to check pairwise compatibility between the new and each device which is already assigned. All devices have a period P_i and an offset O_i, which is the number of minimum periods until it will transmit next time. If there exists integers m and n such that the equation below is true, then the new device cannot fit into offset O_{new} since it will collide with device i:

$$O_i + mP_i = O_{new} + nP_{new}$$

The equation can be rearranged to:

$$mP_i - nP_{new} = O_{new} - O_i$$

For this equation the following property holds, where the equation has an integer solution when:

$$(O_{new} - O_i) \bmod \gcd(P_{new}, P_i) = 0$$

The greatest common divisor of the two device periods represents the finest grain at which overlaps are possible. Thus if the difference in offset is divisible by the GCD then the transmissions will align at some point. The compatibility is checked for all devices in the slot for a given O_{new}, and if one of them results in an offset difference divisible by the GCD, then it is not compatible, and the protocol proceeds to check the compatibility in relation to another slot or another O_{new}. It does not make sense to check compatibility for large values of O_{new} since this would assign the device far into the future. Therefore, this procedure breaks out if an upper-bound threshold is reached.

As described in figure 3.8 the protocol tries to find an unassigned slot if none of the other slots are compatible. But if there are no more unassigned slots, then it proceeds to the last step. Here, a set of available time slots are retrieved, compared and the device is assigned to the time slot with the longest lifetime. The available slots are found using the functionality introduced in section 3.3. Each of the available slots have an offset indicating when they are available from and a collision time for when they will collide with one of the other devices in the time slot. The difference between these two is the lifetime of the time slot assignment and the device will be assigned to the one with the longest lifetime.

3.4.1 Sliding Window Assignment (SWA)

With CPA, devices are assigned into available time slots that are either forever compatible or temporary. When networks become more dense the possibility of finding a compatible slot becomes lower and thus the devices joining the network late will be more likely to be rescheduled. This makes CPA somewhat unfair since rescheduling can result in shifting of the device's period, and frequent rescheduling results in a device period that keeps deviating from its original period.

The SWA introduces an expansion of the rescheduling and assignment procedure to avoid the unfairness of CPA. When a device needs to be rescheduled in SWA, it is not only the available slots that are considered anymore. Instead, some of the devices in the schedule will be removed in order to open up more possibilities for finding compatible slots. To determine which devices that are removed a sliding window is used. The sliding window, which is illustrated in figure 3.9, starts with the current slot and can have its length adjusted. All devices within the window will be removed from the schedule and then assigned to a time slot again. The assignment procedure is very similar to CPA seen in figure 3.8, however, in SWA each step is performed for all devices within the sliding window before moving on to the next step. This means that the procedure first tries to find a compatible slot for all devices but for the ones that cannot find a compatible slot, they will be assigned to the slot with the longest lifetime.

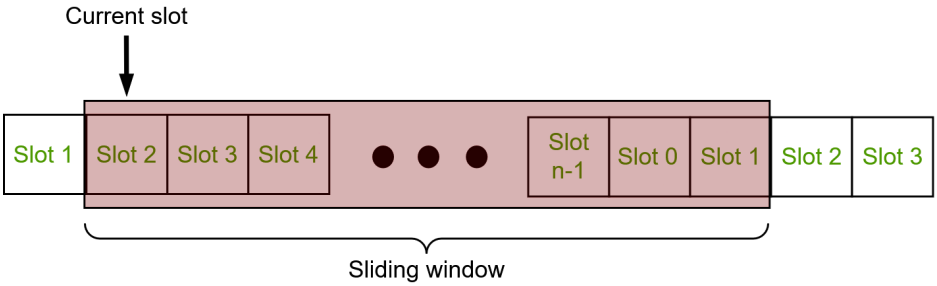


Figure 3.9: All devices which are going to transmit in slots within the sliding window will be considered for rescheduling

After this procedure is finished the server can send a downlink to reschedule the device which triggered procedure. But for the remaining devices which had also been assigned to new slots, the server has to wait for their uplink messages in order to send their rescheduling information. This changes the flow of the server, where in the previous versions the rescheduling was always triggered by an upcoming collision in the schedule. This change in the server's flow can be seen in figure 3.10.

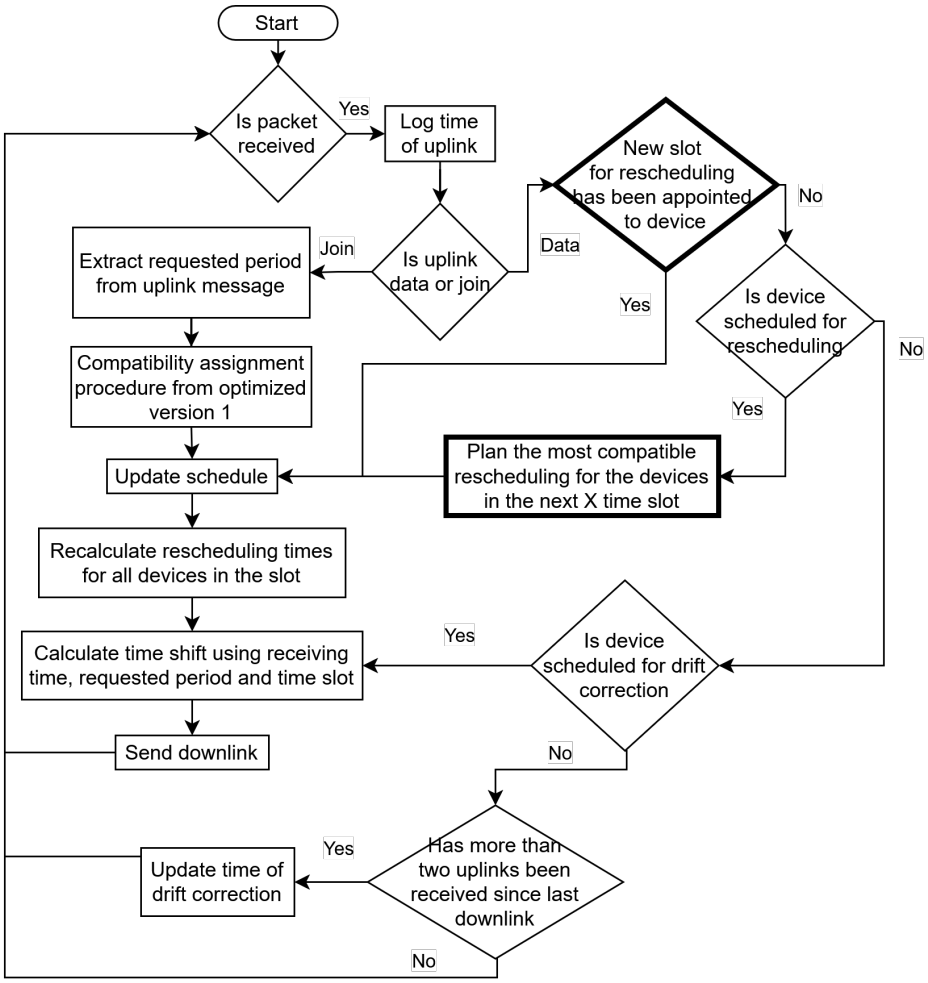


Figure 3.10: Flowchart of SWA. Boxes with fat borders mark the changes from the original flowchart in figure 3.6.

This strategy focuses on distributing rescheduling more evenly instead of placed solely on devices that are joining late. Although this leads to an increased amount of reschedulings, it forces the system to consider more possible schedules, thereby avoiding suboptimal scheduling based on when the devices join.

Chapter 4

Simulator Design

In order to evaluate the performance and scalability of the proposed protocols, a discrete-event simulator has been developed. This was chosen instead of a real world deployment and existing simulators due to the following reasons: A real world deployment would be too expensive and complex for the large-scale scenarios that will be tested. Existing simulators, on the other hand, were found to lack the customizability necessary for generating the desired results. Additionally, many of these tools are computationally heavy, making them unsuitable for testing the scalability of large-scale systems efficiently.

The custom simulator is capable of capturing detailed metrics for different network sizes using each one of the protocols. It is implemented as a discrete-event simulator driven by an event queue, where the simulation clock advances according to the timing of the next event retrieved from the queue. Figure 4.1 shows an overview of the simulator.

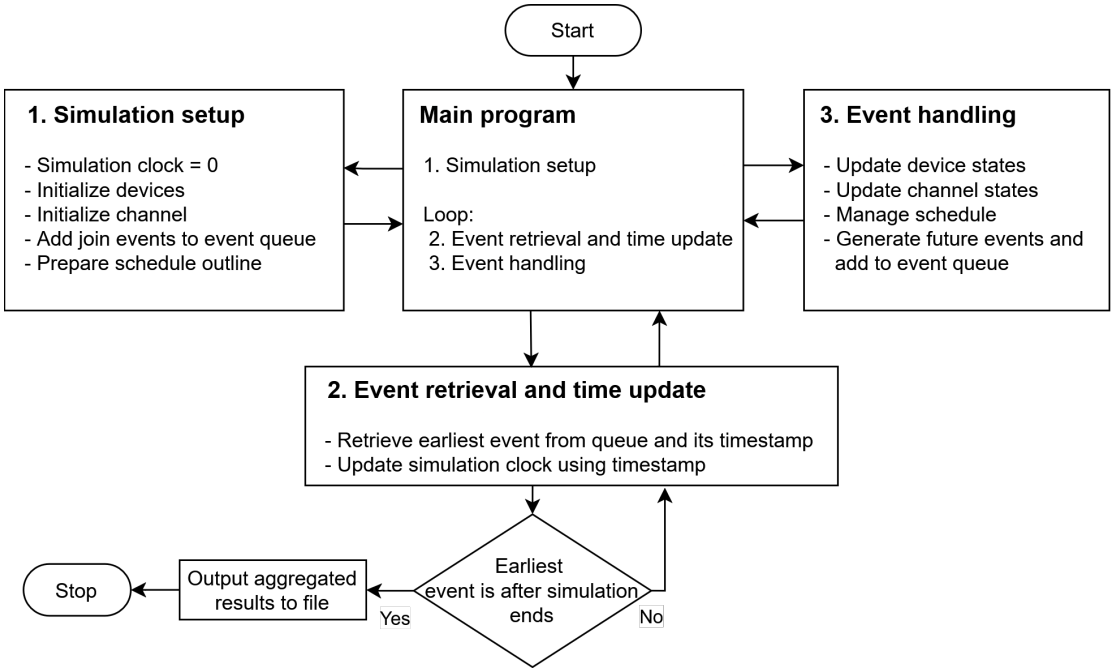


Figure 4.1: Overview of the discrete-event simulator

The code can be found in this Github repository: `https://github.com/SimonJuhl/LoRaCollaborativeMAC`

4.1 Simulator Requirements

The simulator is designed to generate and output the following performance metrics:

- Utilization of the channel
- Energy consumption: Total energy used by each device, accumulated based on time spent in transmit, receive, and sleep states.
- Successful transmissions for each device
- Downlink communication statistics including how many times devices are rescheduled and drift corrected
- Timing shifts which show how much time the devices are required to shift their transmission time when being rescheduled. This is to indicate a device's deviation from its intended periodicity

The utilization metric requires tracking of the channel state in order to calculate the amount of busy time. It also needs to be able to detect collisions in order to differentiate between successful busy time caused by successful transmissions and colliding transmissions.

4.2 Core Components

4.2.1 Event Class

Each event in the simulator is an instance of the event class. It is a simple structure that contains the following attributes:

- Event time in microseconds
- Event type (`TX_START`, `TX_END`, `RX_START`, `RX_END`, `SHORT_RX_START`)
- Device ID
- Presence of collision. If a collision is detected when an uplink or a downlink transmission starts then this attribute is set to `True` for the event that ends that transmission. This will ensure that the packet is dropped.

The five event types are used to drive all changes in the system. These events manage state transitions for devices, maintain the channel state for detecting collisions, and trigger protocol-specific logic. Figure 4.2 shows how the events relate to the uplink and downlink transmissions.

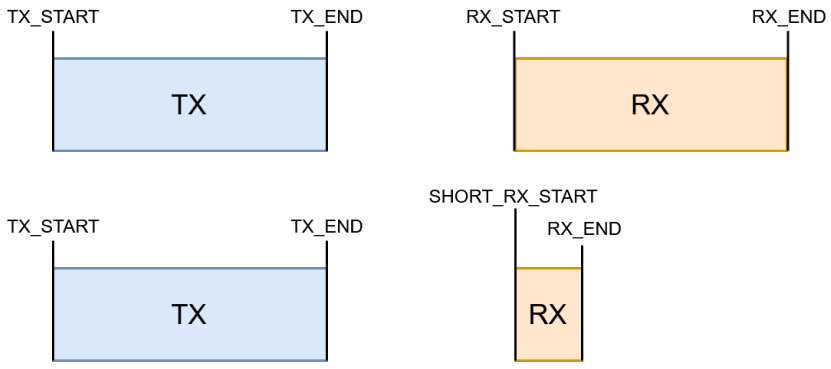


Figure 4.2: Event types

TX_START

This event is triggered when a device begins its uplink transmission. It updates the device's internal state to "transmitting" and signals the channel that a new transmission has started. If the channel already has another active transmission, a collision is detected.

TX_END

This event marks the end of the uplink transmission. The device returns to sleep mode and the transmission is removed from the channel's active transmission list. If there was a collision during this transmission, the packet is marked as failed. If no collision occurred, the network server processes the packet. If this process determines that a downlink is required for rescheduling or drift correction, then an `RX_START` event will be queued up. Otherwise, a `SHORT_RX_START` event is queued up to simulate the device listening for a response.

RX_START

This event is used when the network server has a downlink to send back to the device. It sets the device state to "receiving" and updates the channel state to indicate a downlink is active.

RX_END

This event ends the receive window. If a valid downlink was received, the device reads its content and shifts its transmission time accordingly. The channel object's active downlink is cleared, and the device returns to sleep mode.

SHORT_RX_START

If no downlink is scheduled after a transmission, a shorter receive window is simulated using this event. It changes the device state to "receiving" but does not update the channel state or cause any channel-level behavior. Its only purpose is to account for energy usage from the brief listening period. This event also schedules an `RX_END` event to follow, ensuring the device state is returned to sleep.

In section 4.3.1 an overview of how events are retrieved from the queue and future events are generated will be presented.

4.2.2 Device Class

Each device is modeled as a Python class where all device have their own unique ID. The device's most important attributes are the following:

- States: Time of next transmission, current mode (transmit, receive, sleep).
- Properties: Device period, clock drift, drift direction.
- Metrics: Accumulated energy use, number of successful transmission, number of drift corrections, rescheduling statistics.

All state updates of a device are tied to the events in the simulator. When a `TX_START` event is retrieved from the queue this will change the device's mode from "sleep" to "transmit". The duration of time spent in sleep mode is used to update the energy use of the device.

The `TX_START` event will also update the device's next transmission time using the device's period and the drift which is being modeling. This calculation can be seen below:

$$drift_{\mu s} = \frac{drift_{ppm}}{1.000.000}$$
$$drift_{adjustment} = driftdirection \cdot P_{device} \cdot drift_{\mu s}$$
$$t_{nexttx} = t_{nexttx} + P_{device} + drift_{adjustment}$$

Since the simulator works with microsecond precision, the first line calculates how much drift will happen per second in the unit of microseconds. Second line calculates how much drift this will result in over the duration of the device period. This calculation also takes the drift direction into account which can be either 1 or -1. The direction is given by random to all devices in order to avoid that they drift in the same direction.

4.2.3 Channel Class

The channel is responsible for tracking the number of ongoing transmissions. It has two state attributes which it uses to detect collisions and two attributes for the utilization and collision metric:

- Ongoing transmissions which is being incremented and decremented as transmissions start and end
- Collision detected boolean
- Utilization which is the accumulated uplink transmission time without collisions occurring
- Number of collisions which is being incremented for each collision

The channel's state and metrics are updated whenever a transmission starts or ends. This is handled through the device's method used to change mode, which is called with the current simulation time and the type of event.

When a `TX_START` or `RX_START` event is passed to this method, it checks the current number of ongoing transmissions. If the ongoing transmissions attribute is 1 or more, it means another transmission is already in progress, so a collision is detected. In that case, the collision detected flag is set to `True`. The method returns `True` to signal that a collision has occurred. This return value is used in the event handling logic to determine if the transmission should be considered failed. The collision flag is set to `False` only when the last ongoing transmission ends. This ensures that collisions are only cleared when the channel becomes completely idle again.

When the ongoing transmissions attribute is back to 0 then the utilization is updated if and only if the collision detected boolean is `False`. The simulator is not counting collided transmissions as part of the utilization.

4.3 Scheduling Framework

4.3.1 Event Queue and Time Advancement

At the core of the simulator is the event queue, which holds future actions to be performed. The event types which include starting and ending transmissions and receptions have been described in section 4.2.1. Each event includes a timestamp, a device ID, and an event type. The simulator pops the earliest event from the queue, advances the clock to its timestamp, and handles it. Most event handling functionality is responsible of generating the future events. For instance, the `TX_START` event handling needs to generate the corresponding `TX_END`

event which will end the uplink transmission. The events are generated according to the flow in figure 4.3.

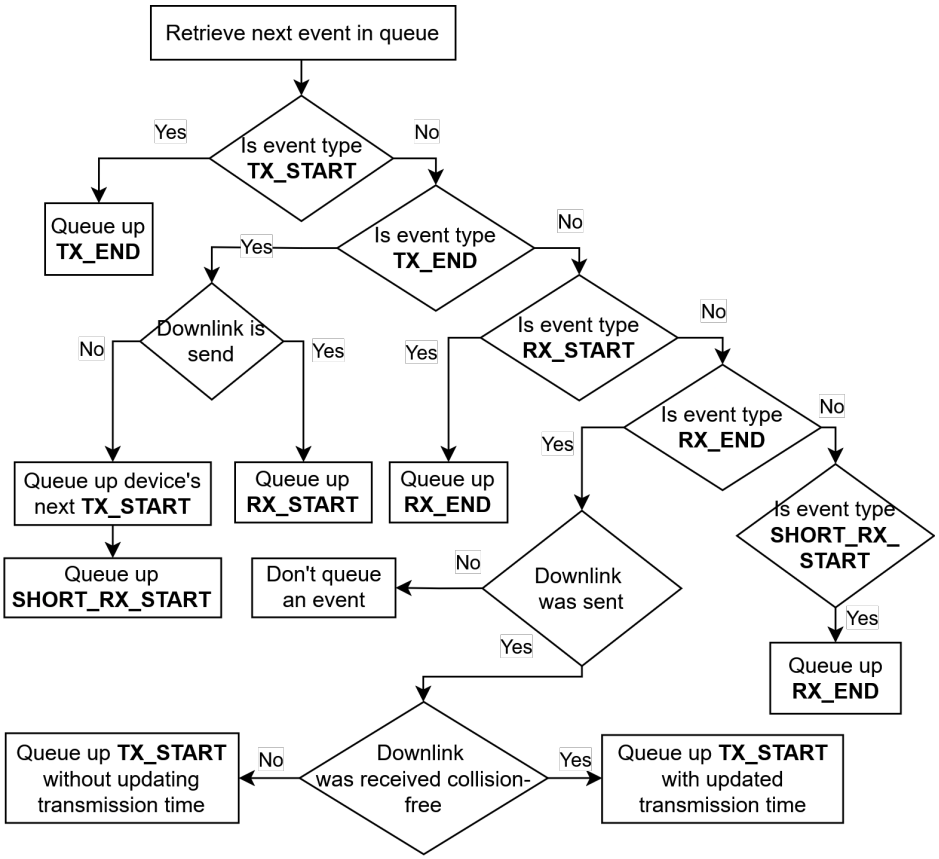


Figure 4.3: How events are retrieved and future events are generated

As it is seen from the figure, the `TX_END` event handler will generate the next `TX_START` event if no downlink message has to be sent. This is because the device's next transmission time does not have to be changed. However, if a downlink message is sent then the `RX_END` event handler will generate the `TX_START` event instead. It will only update the transmission time according to the content of the downlink message if the transmission is received collision-free. Otherwise the transmission time will remain the same without the necessary drift correction or rescheduling time shift.

4.3.2 Slot Assignment Structure

Devices are assigned dynamically when they join the network. The `slot_assignments` structure, seen in figure 4.4, keeps track of which devices are assigned to each slot, each device's transmission period and the current offset of each device. This offset describes the number of minimum periods until a device transmits in the slot next time.

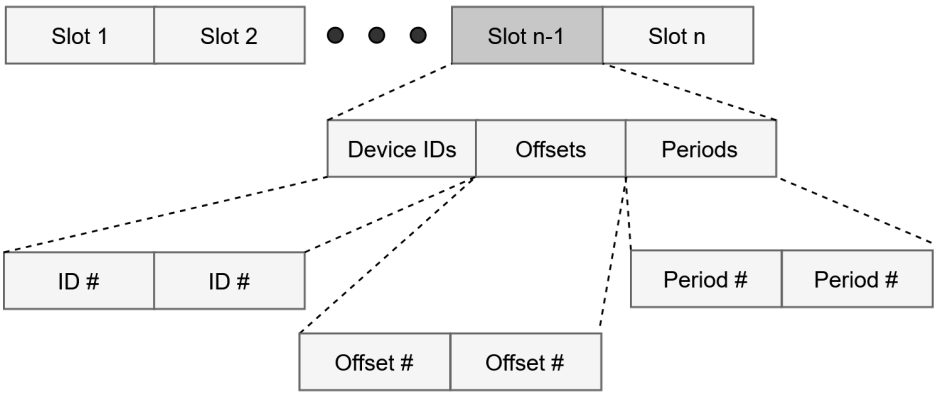


Figure 4.4: Slot assignment structure and how devices are mapped to slots

This structure is updated any time a device joins or is rescheduled. Its nested list format includes all the necessary information which allows the different slot assignment protocols to efficiently check whether a slot is available or compatible for a new device. How the protocols use the `slot_assignments` structure to do this is described in further detail in section 4.5.

4.4 Event Handling

Among all of the event handling procedures, the most comprehensive event handling happens for the `TX_END` event, which marks the ending of an uplink transmission. The protocol design from chapter 3 is the foundation for the functionality of this event handler. In figure 4.5 an overview of the functionality can be seen.

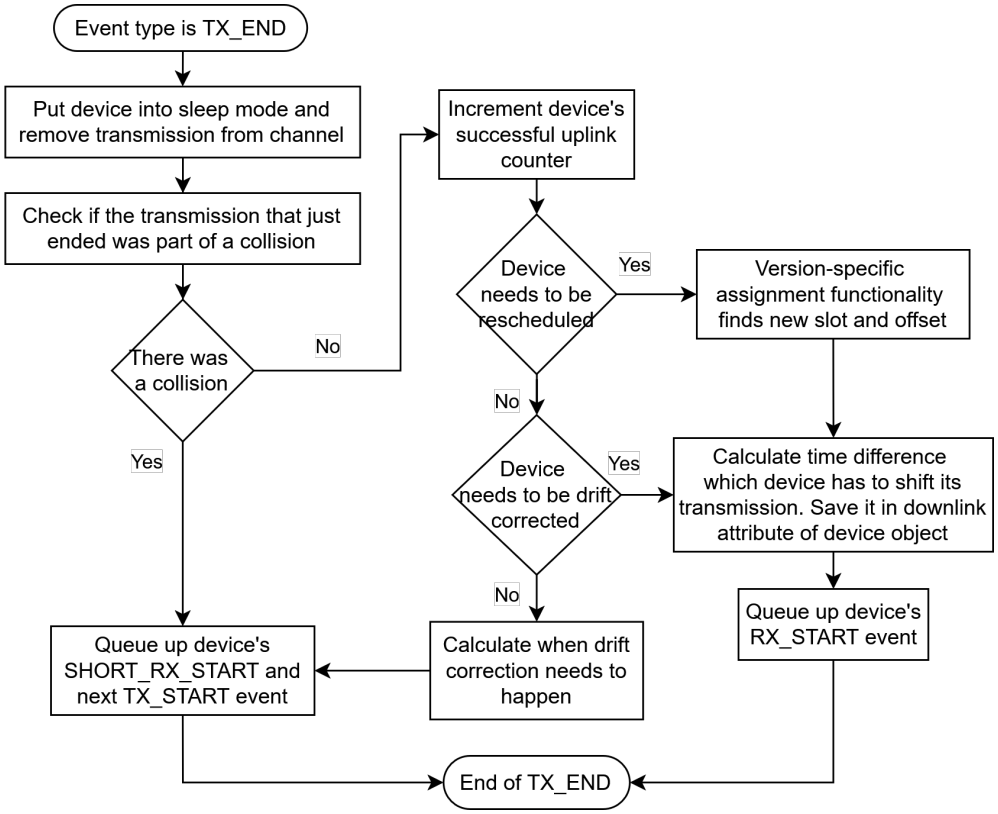


Figure 4.5: Detailed handling of a `TX_END` event

When the device and channel states have been updated then the channel returns a boolean

that tells whether a collision has occurred on either the `TX_START` or `TX_END` event. If it is collision-free then the device's successful transmission metric is updated and it is checked whether a rescheduling needs to happen. If a rescheduling needs to happen then the protocol version's specific functionality is used to find a new slot and offset to use for assigning the device. The `slot_assignments` structure is updated, the time shift is calculated and send to the device.

4.5 Protocol Logic Integration

While most of the simulator is protocol-agnostic, each protocol provides its own logic for how a device is assigned to a slot. This modularity allows for clean comparisons between protocols.

The random version finds a slot that has an offset equal to the devices transmission periodicity. Thus a device with period $x \cdot$ minimum periods will get an offset of x. The slot will be chosen by random from the set of slots $[1, n]$. For the next slot version the `assign_to_first_available_slot` function is used. It uses the `slot_assignments` structure and follows the procedure seen in figure 3.7. The CPA and SWA also uses the `slot_assignments` structure and their respective procedures are described in section 3.4 and 3.4.1.

Chapter 5

Results

5.1 Simulator Configuration

In the following table, specific parameters used in the simulator runs can be seen.

Parameter	**Value**
Simulation run time	3 days
Number of devices (range)	200 - 3600
Minimum period length	5 minutes
Distribution of device periods	100 - 350 minutes
Rescheduling bound	12 hours

Table 5.1: Simulator specific settings used in the testing scenarios.

For the simulation of end devices power consumption, product information for the LoRa SX127x transceiver is used[4]. The power consumption for different modes can be seen in the table below.

Operation mode	**Power consumption**
Sleep mode	0.2 μA
Receive mode (high power)	11.5 mA
Transmit mode (low power)	120 mA

Table 5.2: Power consumption data for differnt modes of operation for the LoRa SX127x transceiver.

However, it is possible to lower the power consumption using an external piece of hardware with a low power precision clock. With such a clock the power consumption would typically be 106 nA [12]

Lastly, since the project is focused on LoRaWAN communication the following values are specified.

- SF12 is used
- Bandwidth is 125 kHz
- For a 24 byte payload the ToA is calculated to $\approx$ 1483 ms which is rounded to 1500 ms

5.2 Performance Metrics

In order to properly compare the different versions as well as evaluate their individual performance, some key metrics are derived from the simulator runs as mentioned in 4.

Since the primary scheduling technique is to shift devices into certain time slots, it is important to know how much devices are shifted on average as well as the maximum and minimum values are to compare if versions are fair or if some devices are disadvantaged. Other important metrics are energy consumption, overall utilization, amount of reschedules, collisions, and overhead communication.

5.3 Performance Evaluation

The primary focus of the project is to enable LoRaWAN networks to scale better by ensuring a more reliable medium access protocol. This means better overall network throughput and higher utilization.

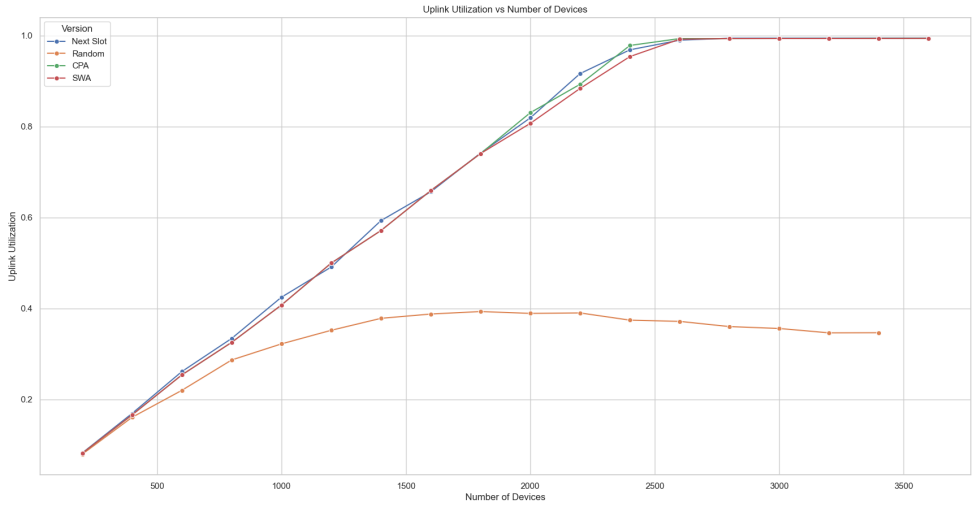


Figure 5.1: Uplink utilization for the 4 versions for different network sizes

It is clear from figure 5.1 that the random version, which plateaus at a utilization of 0.4, have a very low ceiling in terms of supported nodes. Meanwhile, the three proposed protocols manage close to 100 % utilization.

However, having a high utilization alone is not necessarily a good sign, since it does not tell the whole story. Therefore, it is important to include other metrics which can be compared to the other protocols in order to better determine which of them is best in practice.

In figure 5.2 it can be seen how much devices, on average, are shifted for the different versions as well as for different network sizes. As the number of end devices increases, the system needs to shift devices a lot more in time in order to ensure that the transmissions remain collision-free. This is expected behavior since the three protocols NSA, CPA, and SWA all hit a utilization of ≈ 0.994 meaning that there isn't a lot of unused time slots in the schedule. Also, by comparing the plots from figure 5.1 and 5.2, it is clear that when the utilization peaks at 2400 end devices in the system is also the time where the required average rescheduling shift begins to grow linearly.

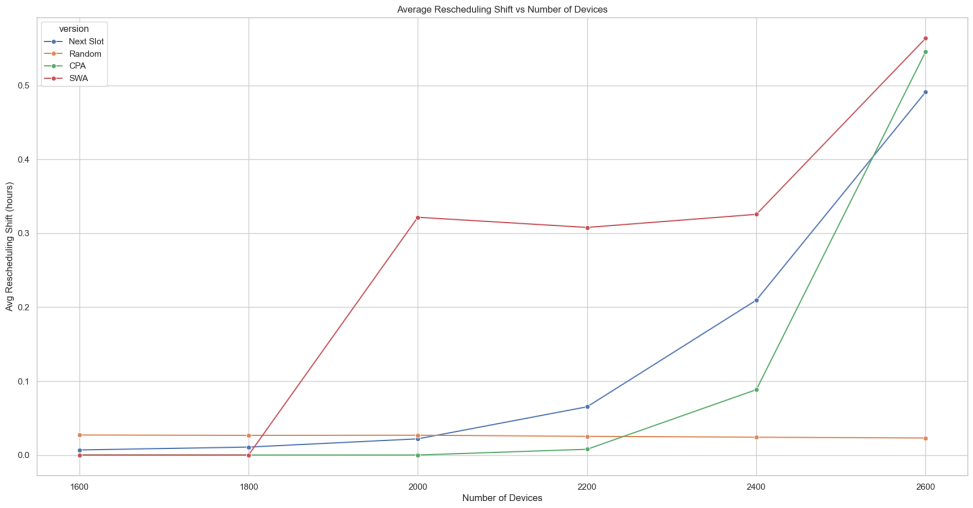


Figure 5.2: The average amount of reschedulings for a specific set of nodes

Now, when inspecting figure 5.2 the Random version seems to perform really well, by keeping a near constant average. However, this is due to the way the Random version operates. Instead of looking for an ideal time slot, it simply reassigns an end device to a random time slot. This means that the protocol does not look long into the future to find the optimal schedule. Another part of what the above figure does not capture is the fact that the Random version experience a lot of collisions when the amount of end devices increase. This can also explain why the utilization drops a little for the Random version, when the number of end devices surpasses 2400. At a network size of 2400 nodes, there are around 27.000 collision over the span of 3 days for the Random version.

Based on the results shown in the figure above regarding average rescheduling shift, two interesting questions arise. Firstly, with the increase in shifts as the number of end devices grows the availability decreases, so how does this trade off impact the availability of the devices. The second question builds on top of this and focuses on the fairness of the protocol, since some protocols utilize the concept of compatible devices, which can coexist in the same time slot without interfering. This means that some devices may get rescheduled over and over again, while other devices never miss a transmission or experience any other form of delay in its transmission.

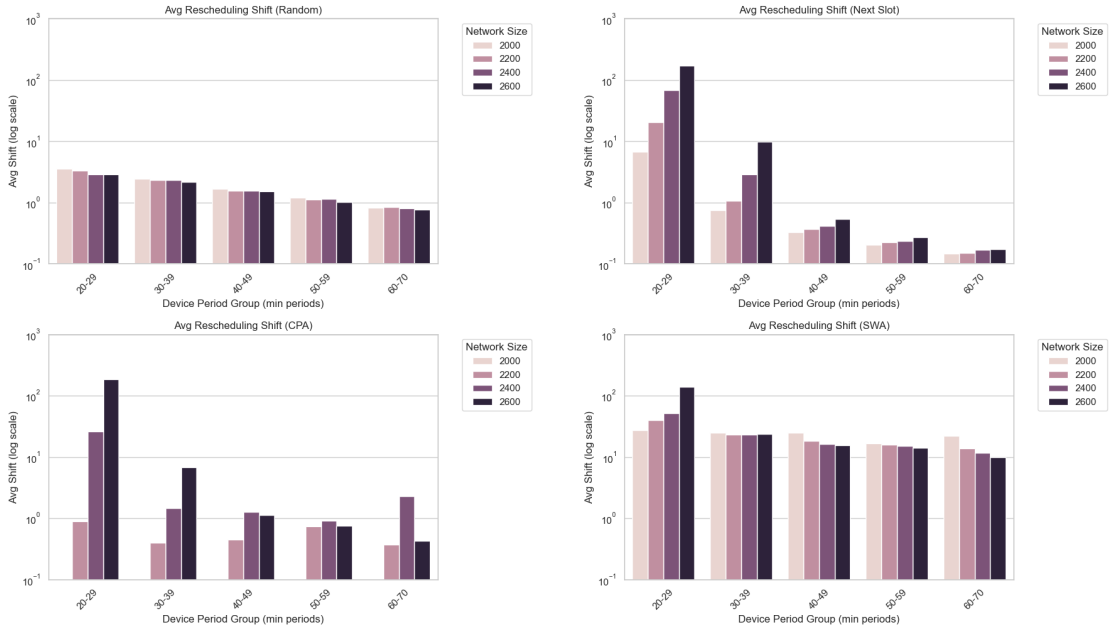


Figure 5.3: Plot of how much devices with certain periodicities are shifted on average for different assignment versions as well as network configuration.

Figure 5.3 shows a plot of the 4 different versions evaluated for different number of end devices. For each number of end devices the average shift is plotted, however, the results are grouped by the end devices' periods to see how the average shift is distributed across the different periods. As seen in the figure, the Random version quite evenly distributes the shift across all periods, where the other versions seem to favor devices with longer periods. Logically, this makes sense because an end device with a short period takes up more time slots over a given amount of time compared to a device with a longer period. I.e. this means that the devices with lower periods are harder to fit into a schedule where they can stay for a long time. For the two optimized versions, it is possible to create a schedule such that no reschedules are necessary up until 2200 end devices simply by using compatible time slots.

By counting the amount of reschedulings per period and the amount of successful transmissions per period, it is possible to calculate the probability of a given period experiencing rescheduling. These probabilities are calculated and plotted for the grouped periods in order to examine how the protocols affect different devices based on their periodicity.

The probabilities can be seen in figure 5.4.

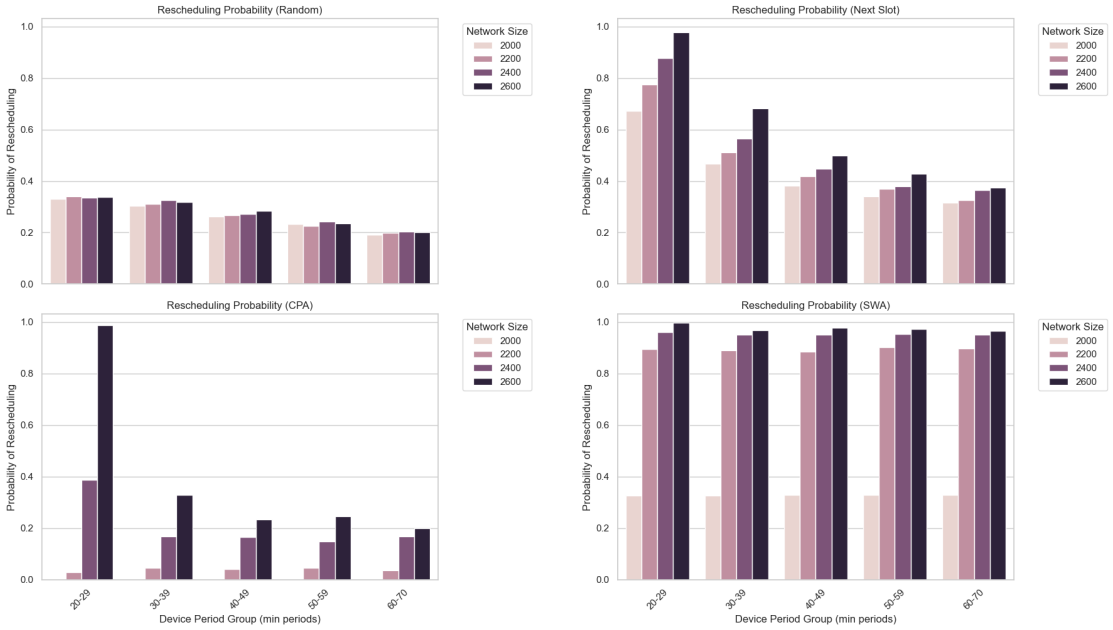


Figure 5.4: The probability of a device with a certain period being rescheduled during the simulation.

It is immediately apparent that the shorter periods are being unfairly treated compared to the longer periods for CPA and NSA when the network grows in size just as discussed before. Focusing on the periods between 20 and 29, it can be seen that the probability increases rapidly for the CPA when the network size increases. However, the same is not the case for the longer periods, there the increase in probability is much lower. The same can be observed in the plot for the NSA version. This indicates that the proposed protocols CPA and NSA scale quite poorly if measured against fairness in rescheduling. A key difference in CPA and NSA is that for smaller network sizes, CPA seems to be quite fair in terms of rescheduling regardless of periodicity compared to NSA, but when the network is scaled the same problem arises with CPA as is apparent with NSA, which is the unfairness towards short periods.

SWA on the other hand have very equal probabilities across all periods and for all network sizes. Since SWA takes all nodes into consideration, compatible and non-compatible devices, when rescheduling, compatible devices may be rescheduled to make way for more optimal schedules.

An important metric to measure is how well the different versions support the periodicity of individual devices when the network size increases. Since the average amount of shifting will increase, there may be scenarios in which some devices shift so much that they miss one or more transmission periods, suddenly failing to uphold their periodicity. Looking at the short-term availability of the devices, some may be unavailable simply due to being shifted more than their own period in a single rescheduling. Since SWA may shift devices quite a lot in order to ensure a collision-free schedule figure 5.5 has been created. It shows how the shifts are distributed for SWA at a network size of 2000 end devices as well as indicating when devices have been shifted for more than their own periodicity leading them to become unavailable.

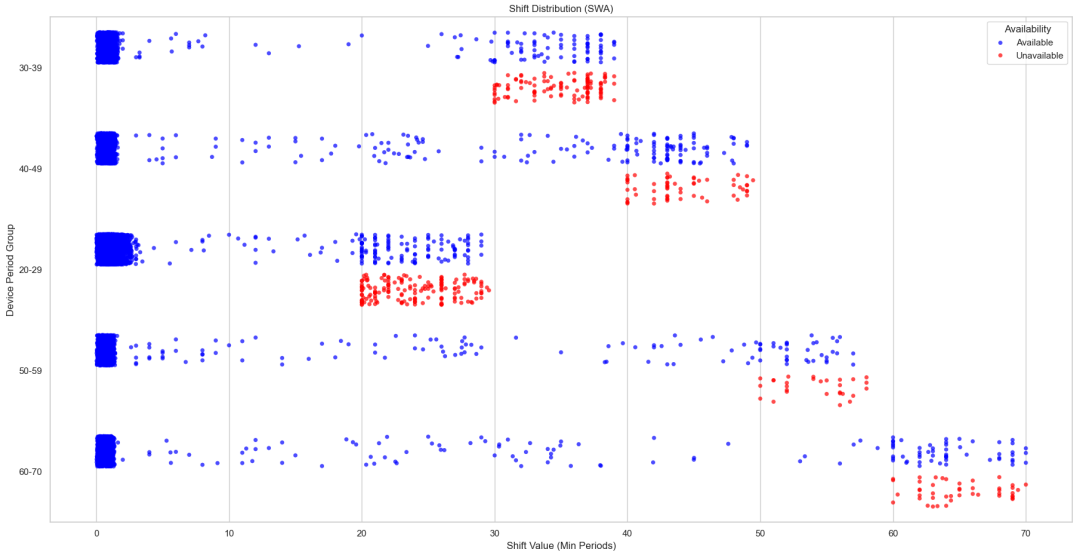


Figure 5.5: Drift distribution of SWA for 2000 devices coupled with short-term unavailability of devices

Figure 5.5 clearly shows that even though SWA is capable of finding a schedule that fits all 2000 devices, a large part of them experience short-term unavailability simply due to missing periodic transmissions due to large shifts produced by the assignment method.

As discussed in chapter 3 the downlink transmissions consists of two different message types, drift correction messages and rescheduling messages. Both message types are important for the operation of the overall system and the ratio between the two message types can be seen in figure 5.6.

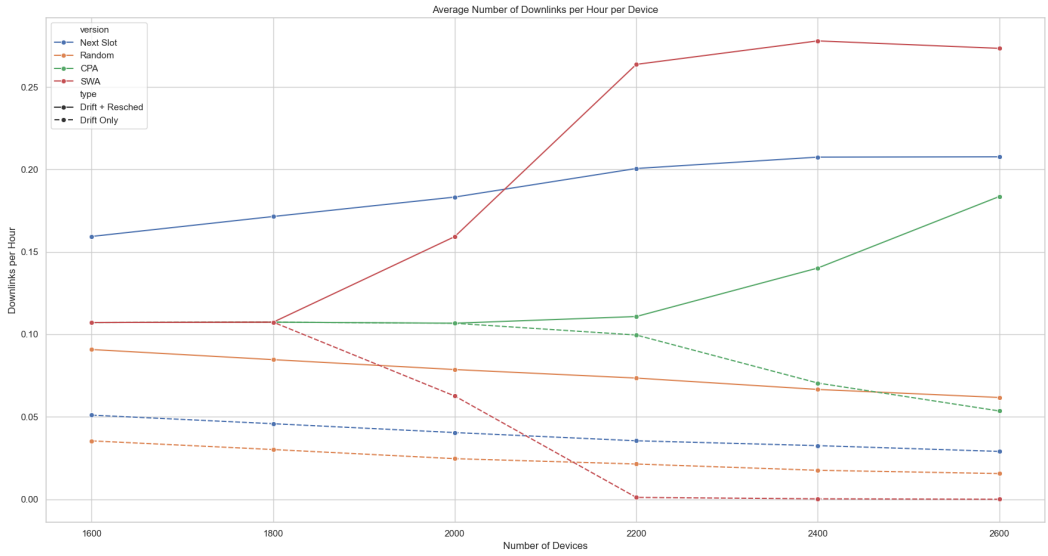


Figure 5.6: The average number of downlinks being transmitted

The solid lines show the average number of downlink messages and the dashed lines show the average number of downlink messages that are drift correction messages. As clearly illustrated in the figure, the number of donwlink messages used for drift correction is decreasing with the increase of reschedulings. The reason behind this is, that the network server can

calculate at every uplink the devices misalignment with the time slot, it is possible to account for the misalignment when rescheduling the end device. So if a device needs rescheduling before it needs drift correction, both can be done at the same time. This limits the aggregated number of downlink communication in the system, thereby limiting the power consumption.

As the utilization increases, more and more end devices needs to be rescheduled often since there are very few time slots that are free. However, there are some noticeable differences between the 3 collision-free versions, NSA, CPA, and SWA. While the NSA and CPA versions sees a drop in drift correction messages, there are still drift correction since some of the devices that have been scheduled is in compatible slots meaning that it can transmit forever without experiencing collisions given that the drift is corrected. However, for the SWA the average amount of drift corrections become almost 0, with only 3 drift corrections being made for the entire simulation at a network of 2800 end devices compared to the 9019 drift corrections being made for a network of 2000 end devices. This fact and figure 5.6 in its entirety highlights the differences in CPA and SWA. SWA was designed with the intention that when end devices join the network in an early stage they are assigned a compatible time slot and generally kept there, however, there could be a more optimal schedule later in time when new devices are joining. With this in mind, SWA was designed such that it tries to find optimal schedules by looking at multiple devices at the same time and seeing if a better schedule can be achieved. This procedure of rescheduling multiple devices in order to find better schedules leads to, for large networks, almost all end devices being rescheduled before the drift even becomes a problem, which was also seen before in the descussion of rescheduling probabilities for different device periods. This large difference in operation can be seen quantified in table 5.3, where the number of drift correction messages and the number of rescheduling messages can be seen for different network sizes.

Network size	CPA		SWA	
	Drift correction	Rescheduling	Drift correction	Rescheduling
2000	15.370	7	9019	13.918
2400	12.185	12.046	52	47.976
2800	8.480	30.182	3	51.771

Table 5.3: Amount of each downlink message type for the protocols CPA and SWA for different network sizes

As the values in table show, there are clear indications that the two versions operate quite differently in terms of rescheduling devices to ensure collision free transmissions. CPA is capable of supporting 2000 nodes while only having to do rescheduling 7 times, SWA requires 13.918 reschedulings to do the same, but does however not drift correct as often as a consequence of the reschedulings. In total however, for 2000 end devices, SWA requires about 7500 extra downlink messages to keep the schedule collision free compared to CPA.

Interestingly enough, the higher number of downlink messages needed for running SWA does not lead to substantial differences in energy consumption between CPA and SWA.

In figure 5.7 below, the energy consumption pr. successful transmission is shown. By calculating the total energy consumption and dividing it with the number of successful transmissions gives an indication of how energy efficient the assignment protocol is. If the number of successful transmissions is quite low, as is the case with the random assignment, more power has been wasted trying to transmit the same amount of information.

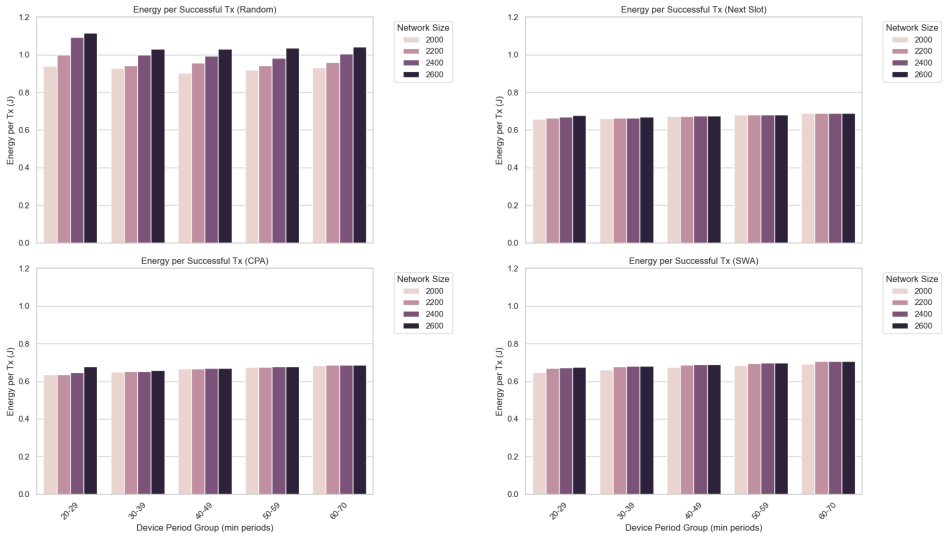


Figure 5.7: Energy consumption by number of successful transmissions

As seen in the figure, the energy consumption is nearly identical for the three proposed assignment protocols across all periods. The difference is minimal, which indicates that even with the additional amount of downlink messages in SWA the power used when receiving messages is negligible compared to the power usage of transmitting uplink messages.

5.4 Stress Testing the Protocol

As mentioned previously, the scheduling methods rely on the rescheduling bound, which is a metric for how long the system should be able to operate before needing to be drift corrected. This rescheduling bound also sets a boundary on what the largest period can be. If this boundary is not uphold, the system cannot ensure collision-free transmissions. In figure 5.8, it can be seen that merely by changing the periods of devices the system operate quite differently. This is due to the system not being able to handle drift corrections for devices with periods greater than half of the rescheduling bound because it cannot estimate the drift before it happens, which leads to drifts in the system that cause collisions.

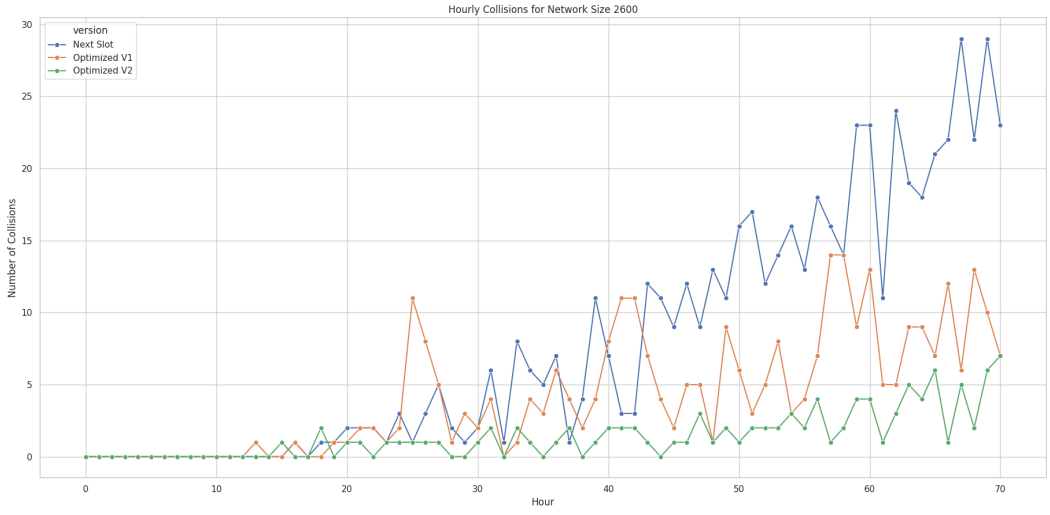


Figure 5.8: Number of collisions for the different versions when periods range from 20-100 instead of 20-70

5.5 Summary of Results

The results reveal different behaviors of the protocols as the network scales from sparse to dense. In sparser networks, most devices can be scheduled without conflict using compatibility-based strategies, as their transmission periods align with available slots. This is where protocols like CPA and SWA perform particularly well, as they prioritize finding compatible assignments that can last indefinitely without needing rescheduling.

However, as the network grows denser, the assumptions of compatibility break down. Perfect period alignment becomes rarer, and rescheduling becomes a necessary part of maintaining a collision-free schedule. At this point, the performance and strategy of each protocol begin to diverge. NSA and CPA tend to maintain tighter control over slot occupancy but may begin to exhibit unfairness, where certain devices, especially those with shorter periods, are rescheduled more frequently. SWA, on the other hand, responds to increasing density by redistributing devices more aggressively, which improves fairness but at the cost of more frequent rescheduling and greater downlink overhead.

These dynamics highlight the tension between schedule stability and fairness as the system scales. The scheduling protocols must not only prevent collisions, but also manage how rescheduling is distributed across devices. This suggests that in dense scenarios, further optimization may be needed to better balance fairness, shift magnitude, and overhead.

Chapter 6

Limitations of the current evaluation method

6.1 Trade-Off Between Throughput and Drift Tolerance

A central design consideration in the proposed scheduling protocols is the use of a guard interval to absorb timing inaccuracies caused by clock drift. The guard interval ensures that transmissions do not overlap even as devices gradually drift from their scheduled time slot. However, this robustness comes at the cost of efficiency. A longer guard interval reduces the number of usable time slots in a fixed-length period, which in turn lowers the number of devices that can be scheduled concurrently. Conversely, a smaller guard interval increases capacity but reduces tolerance to drift, especially for devices with long transmission periods.

This trade-off between throughput and drift tolerance creates a upper bound on how many devices can be supported in a schedule given a target period length and clock drift specification. In this project, this trade-off is balanced through the definition of a rescheduling bound, which limits how long a device can drift before the server must intervene with a correction. Still, this trade-off is not imposed by hardware and time limitations alone, it is a direct consequence of the protocol's requirement to remain collision-free. If the system was allowed to tolerate occasional collisions, it could operate with shorter guard intervals and more relaxed rescheduling bounds. This would allow support for a wider range of device periods. However, this relaxation would come at the cost of network stability. As collisions accumulate, throughput would degrade and the reliability of scheduled communication would suffer. In short, the protocol favors predictable and reliable operation over maximizing slot utilization, accepting stricter timing constraints in order to avoid the cascading effects of packet loss.

6.2 Improvement of Simulator's Realistic Behavior

The simulator created for this project is based on some assumptions and simplifications in order to focus on the scheduling aspect of the problem. However, these simplifications leads to unrealistic behavior in the simulator. Specifically, the way collisions are handled are simplified compared to real lift scenarios. As described in 2.4 there are a number of outcomes that may occur when to LoRa signals collide, which depend on when the two signals collide. In some cases one of the signals can be retrieved and the packet can be extracted while other at other times, both packets are lost. In order to ensure a higher level of realism in the simulator, there would have to be added received signal strength which would be modeled based on the power output and the geographic location of the transmitting device. The

signal strength is a key metric in terms of realistically simulating the capture effect and other collisions that may occur in LoRa-based networks.

The simulator also takes place in a vacuum in terms of other signals. If the solution should operate in densely populated areas it would be reasonable to assume that, there could also occur interference between different spreading factors, inter-spreading factor interference. While the spreading factors used in LoRa are designed to coexist, they are not entirely orthogonal and therefore also not entirely free of interfering. Therefore, the simulator could also be improved such that it is capable of simulating other networks transmitting on different spreading factors and on the basis of these new devices' power outputs and locations calculate the Signal-to-Interference Ratio (SIR). This metric can be used to compare to the SIR threshold in order to determine wether or not a received signal is decodable given som level of inter-spreading factor interference. This problem however, is based on co-channel inter-spreading factor interference and can be solved by utilizing multiple channels.

Chapter 7

Conclusion

This chapter summarizes the key findings and insights from the design and evaluation of server-driven scheduling protocols for LoRaWAN, with a focus on reducing collisions while maintaining scalability and robustness to device variability as presented in the problem statement:

How can a LoRaWAN network reduce collisions through server-driven scheduling, while remaining scalable, and robust to variation in device periodicity and drift?

The evaluated protocols, NSA, CPA, and SWA, demonstrate clear advantages over random-based access which is similar to ALOHA-based access. While the proposed protocols maintain near 100% utilization and significantly reduce the number of collisions even at high device densities. The random assignment version quickly degrades under load, plateauing at $\sim 40\%$ utilization and accumulating thousands of collisions in dense scenarios.

These results strongly indicate that time-based scheduling is an effective strategy for improving LoRaWAN network performance. The use of structured time slots coordinated by the network server enables reliable communication for thousands of devices, provided that devices follow known periodicities and that rescheduling is applied when necessary. This supports the conclusion that time-based allocation is a viable alternative to ALOHA in dense, periodic LoRaWAN deployments.

7.1 Further Improvements

7.1.1 Further Development of SWA

The motivation behind creating SWA was to develop an approach capable of avoiding sub-optimal scheduling by allowing multiple devices to be rescheduled at the same time. Since devices join at random times, the network server is not capable of assigning all devices optimally to minimize or eliminate rescheduling from the outset. The idea was that by rescheduling batches of devices, the system could gradually converge toward a more optimal schedule with minimal future rescheduling.

However, in practice, SWA does not reach such an optimal state. Instead, the sliding window approach often causes excessive rescheduling by including all devices in the window, even those that do not require rescheduling. This leads to high overhead and increased disruption across the network.

While SWA reschedules devices more evenly across different period lengths, avoiding the clear bias seen in CPA and NSA toward rescheduling short-period devices, the overall *rescheduling probability remains high for all devices.* This means that although fairness improves in a

technical sense, it does so by subjecting more devices to frequent disruption, which is not a desirable trade-off.

To address these limitations, future versions of SWA should aim to keep the core idea of batch rescheduling, but implement it in a more selective and context-aware way. Several refinements could improve performance and reduce overhead:

- Track rescheduling history: deprioritize devices that have already experienced significant time shifts to avoid repeatedly penalizing the same devices.
- Avoid large time shifts: do not reschedule devices if the only available slots require major timing changes, as this harms availability and predictability.
- Leverage existing downlinks: prioritize devices that already require a downlink for rescheduling or drift correction, bundling additional rescheduling with already-needed communication.
- Use compatibility scoring: find way of evaluating how stable and compatible a schedule is and whether a batch rescheduling will increase of decrease the stability. If a schedule has a high score, rescheduling may be avoided unless a better configuration can be found.

Thus, a more nuanced approach which incorporates the above mentioned measures could retain the benefits of batch adjustments while reducing unnecessary disruption and communication overhead.

7.1.2 Improved Time Utilization with Dynamic Slot Structuring

One underutilized components of the current time slot structure is the downlink section. Each time slot typically reserves 2.5 seconds for downlink communication, where one second is delay after the uplink, followed by a 1.5-second receive window, regardless of whether a downlink message is actually needed. This design ensures reliable downlink delivery but results in idle time particularly when a stable schedule can be obtained where rescheduling is not necessary. If this unused time could instead be used for additional uplinks, overall throughput could be significantly improved. Alternatively, the freed space could be used to increase guard intervals and improve drift tolerance.

However, using the downlink portion of a time slot for additional uplink transmissions introduces a challenge. If a device is scheduled to transmit during what would have been another device's downlink window, it cannot receive any downlink messages itself at that time. This becomes a problem if the device later needs to be rescheduled or drift corrected, since it would require a downlink message to do so. But because its slot was designed without a downlink section, there's no opportunity to send that correction—effectively trapping the device in a slot where it cannot be updated.

To address this, the slot design would need to become more flexible. Rather than assigning every slot both an uplink and a downlink phase, the schedule could differentiate between:

- Uplink-only slots, optimized for throughput
- Full slots, reserved for devices that are require drift correction or rescheduling

This would allow the network server to pack transmissions more tightly while still ensuring that necessary downlink messages can be delivered. For example, a device could transmit in two uplink-only slots and then be assigned a full slot on its third transmission, when a drift correction is due.

A challenge in implementing this approach is that the server must take into account the future drift correction and rescheduling needs of all devices when constructing the schedule. Since a full slot takes up more time than an uplink-only slot, using a full slot shifts subsequent slots forward and potentially affects the periodicity and alignment of many devices.

To mitigate these issues, the network could define a fixed number of full slots within each minimum period. This would make the period predictable, even if some slots are longer. The server's task would then be to assign devices strategically so that only a subset receive corrections or rescheduling in each period, aligned with the available full slots.

Another design challenge is determining the right ratio of uplink-only to full slots. Too few full slots could cause drift correction or rescheduling opportunities to be missed, leading to collisions. Too many would reduce the potential for high utilization. Finding this balance would require profiling real device behavior and possibly adapting the ratio dynamically as network conditions evolve.

Even with these measures in place, a key challenge remains: ensuring that device periodicities align with time slots, especially when the schedule includes a mix of uplink-only and full slots. Since full slots occupy more time, the position of each slot within the minimum period can shift depending on how many full slots are inserted before them. This changes the starting times of slots within the minimum period, which risks creating misalignments with device transmissions.

One way to address this is to inform devices of these shifts explicitly. However, this would increase communication overhead and require the server to predict and maintain the schedule far in advance. A better alternative may be to impose a consistent structure or pattern in the placement of uplink-only and full slots so that, while slot positions shift from one minimum period to the next, they realign with each device's transmission interval over the course of each device's full period. In other words, although the slot timing may vary from one minimum period to the next, it still aligns correctly with the device's periodic schedule.

However, designing a structure that can consistently handle the dynamics and guarantee predictable alignment will require further investigation.

Bibliography

[1] TheThingsNetwork.org. *LoRa Physical Layer Packet Format*. [Online; accessed 1. March 2025]. URL: https://www.thethingsnetwork.org/docs/lorawan/lora-phy-format/.

[2] The Things Network. *Spreading Factors*.

[3] The Things Network. *Forward Error Correction and Code Rate*.

[4] Semtech. *SX1276/77/78/79*. [Online; accessed 6. March 2025]. Dec. 2017. URL: https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf?srsltid=AfmBOopbePM1fXc5a_SL-3-JMk46SHQb8OVyPLkNIXSCsCQMFIsFYD1Z.

[5] Espressif Systems. *ESP32S2 Hardware Design Guidelines*. [Online; accessed 28. May 2025]. URL: https://cdn.sparkfun.com/assets/5/2/8/9/4/esp32-s2_hardware_design_guidelines.pdf.

[6] Daniele Croce, Michele Gucciardo, Stefano Mangione, Giuseppe Santaromita, Ilenia Tinnirello. "Impact of Spreading Factor Imperfect Orthogonality in LoRa Communications". In: *Communications in Computer and Information Science* (Sept. 2017).

[7] Arshad Farhad, Dae-Ho Kim, Pranesh Sthapit, Jae-Young Pyun. "Interference-Aware Spreading Factor Assignment Scheme for the Massive LoRaWAN Network". In: (Jan. 2019).

[8] Pascal Berthou Laurent Chasserat Nicola Accettura. "Experimental throughput models for LoRa networks with capture effect". In: October (May 2022). DOI: 10.1109/WiMob55322.2022.9941715.

[9] Fernando Kuipers Andri Rahmadhani. "When LoRaWAN Frames Collide". In: (Oct. 2018).

[10] Muhammad Omer Farooq, Dirk Pesch. "A Search into a Suitable Channel Access Control Protocol for LoRa-Based Networks". In: *2018 IEEE 43rd Conference on Local Computer Networks (LCN)* (2018).

[11] Jetmir Haxhibeqiri, Ingrid Moerman, Jeroen Hoebeke. "Low Overhead Scheduling of LoRa Transmissions for Improved Scalability". In: *IEEE Internet of Things Journal* (2018).

[12] NXP. *PCA2131 Nano-power highly accurate RTC with integrated quartz crystal for automotive applications*.