Modelling the energy consumption for over-the-air software updates in LPWAN networks: SigFox, LoRa and IEEE 802.15.4g.

Peter Ruckebusch[1,*], Spilios Giannoulis, Ingrid Moerman, Jeroen Hoebeke and Eli De Poorter

Department of Information Technology
IDLAB - Ghent University - IMEC
Ghent, Belgium
E-mail: first.last@ugent.be

Abstract

The recent trend towards the use of low-power wide-area-networks (LPWAN) communication technologies in the Internet of Things such as SigFox, Lora and Weightless gives rise to promising applications in smart grids, smart city, smart logistics, etc. where tens of thousands of sensors in a large area are connected to a single gateway. However, to manage such a sheer number of deployed devices, solutions to provide over-the-air firmware updates are required. This paper analyses the feasibility of over-the-air (partial) software updates for three LPWAN technologies (LoRa, SigFox and IEEE-802.15.4g) and discusses the best suited update method for different scenarios: full system updates, application updates and network stack updates.

The results indicate that full firmware upgrades consume a substantial amount of energy, especially for the lowest bit-rate LPWAN technologies such as SigFox which drains a single AA battery with 2% when performing a version update. However, technologies with a similar range (i.e. LoRa SF12) require only 0.12%. The trade-off between range and energy (or bit-rate) becomes clear when considering that the least sensitive technology (IEEE-802.15.4g-OFDM) consumes only 0.0001%. Partial updates require significantly less energy for all technolo-

*Corresponding author

gies. Adding a single application uses 6 to 38 times less energy compared to a firmware update, depending on the update method and LPWAN technology. Even partial network stack updates (i.e. MAC) cost 3 to 8 times less energy, making over-the-air updates feasible.

Keywords: LPWAN; Internet-of-Things; partial over-the-air software updates; network management; SigFox; LoRa; IEEE-802.15.4g

---

1. Introduction

In recent years low-power wide-area-networks (LPWAN) such as NB-IoT LoRa, SigFox and IEEE-802.15.4g gained increasing interest from industry and the scientific community in fields related to the Internet-of-Things (IoT). The promise of providing large coverage for low power devices is a key enabler for many use cases in application domains such as smart grids, smart city, smart logistics, etc. because a single LPWAN gateway can serve thousands of sensors within a range of several kilometres. To this end, most LPWANs operate in the sub-1 GHz frequency bands and therefore experience less attenuation and multipath fading.

Although the increased range of LPWAN technologies is appealing for many use cases, LPWAN technologies also have disadvantages. (i) Firstly, they achieve a longer range by using more energy per transmitted bit. The coverage of LPWAN devices is increased by using a lower modulation rate, effectively putting more energy in each transmitted bit (or symbol), thereby resulting in a higher link budget. (ii) Secondly, low power operation is achieved by using a simple star topology, applying an ultra low radio duty cycle, and using a simple, non-synchronised lightweight medium access control protocol such as slotted aloha. As a result, most LPWAN devices only listen sporadically for downlink messages, in most cases just after an uplink transmission.

The aforementioned LPWAN constraints, i.e. the low data rate and low complexity, limit the potential for over-the-air reconfiguration and updates of LPWAN devices. For this reason, most current LPWAN deployments focus

on enabling an uplink for thousands of simple sensors, allowing them to report sensor readings at relatively low reporting intervals. Network management functionality of pervasive IoT networks consisting of constrained devices is still limited. However, it can be argued that over-the-air (partial) software updates are essential for the long-term sustainability and security of deployed networks, especially since in many cases the cost for a manual intervention is a multitude of the cost of the device itself. Post deployment software updates also allow early roll-outs and shorter time-to-market since applications, services or network protocols can be added afterwards.

This paper analysis the feasibility of over-the-air (OTA) software updates in LPWAN networks. More specifically, the cost for over-the-air updates in terms of energy consumption for different technologies (SigFox, LoRa and IEEE 802.15.4g) and radio configurations are determined and compared.

The remainder of this paper is structured as follows. First, Section 2 gives an overview of related work. Next, Section 3 provides a mathematical energy consumption model that is applicable to multiple LPWAN technologies. Section 4 discusses the overhead of different over-the-air software update approaches. Afterwards, in Section 5, the energy consumption models are applied to these different software update approaches. Finally, Section 6 concludes the paper.

2. Background

2.1. LPWAN Technology Overview

There exist a wide variety of proprietary and standardised LPWAN technologies adopting different modulation and coding schemes (MCS). This section gives a high level summary (a more detailed overview can be found in e.g. [1]). Generally speaking, the different PHYs used in LPWANs can be classified as (ultra) narrow-band, spread spectrum or OFDM-based. Table 1 gives an overview of multiple LPWAN technologies for each class.

Table 1: Overview of LPWAN technologies. There are three classes: a) (ultra) narrowband; b) spread spectrum; c) OFDM.

Technology	**Narrow-band**		**Spread-spectrum**				**OFDM-based**		
	SigFox	**IEEE-802.15.4g**	**Weightless**	**LoRa**	**IEEE-802.15.4g**	**INGENU**	**IEEE-802.15.4g**	**IEEE-802.11ah**	**NB-IOT**
MCS	UL: DBPSK DL: GFSK	MR-FSK	GMSK offset-QPSK	CSS	DSSS MR-OQPSK	DSSS UL: RPMA DL: CDMA	B/Q-PSK 16-QAM	B/Q-PSK 16/64/256-QAM	UL: $\pi/2$-BPSK or $\pi/4$-QPSK DL: QPSK
Bandwith	100 Hz	12.5 kHz	DL: 50/100 kHz UL: 12.5/100 kHz	125 kHz	2 MHz	1 MHz	200 - 1200 kHz	1-16 MHz	180 kHz
Band	sub-GHz ISM	sub-GHz & 2.4 Ghz ISM	sub-GHz ISM	sub-GHz ISM	sub-GHz & 2.4 Ghz ISM	2.4GHz ISM	sub-GHz & 2.4 Ghz ISM	sub-GHz ISM	licensed
FEC	N	Y	Y	Y	Y	Y	Y	Y	Y
MTU	UL: 12B DL: 8B	2kB	255B	250B	2kB	10kB	2kB	7991B	UL: 1000B DL: 680B
Datarate (kb/s)	UL: 0.1 DL 0.6	50-200	DL: 3.125-100 UL: 0.625 - 100	0.3-37.5	6.25-50	UL: 78 DL: 19.5	50-800	150 - 346666.7	UL: 250 DL: 226.7

2.1.1. Narrow-band

Narrow-band modulation techniques encode the signal in a low bandwidth in order to obtain a higher link budget because the noise level, experienced inside a single narrow-band, is minimal. Decoding a signal therefore does not require processing gain through frequency de-spreading resulting in a simpler transceiver design. Spectrum efficiency is also high because each carrier only occupies a very narrow-band. The IEEE-802.15.4g standard [2] is a typical example of narrow-band modulation, using a 12.5 kHz bandwidth.

Some LPWAN technologies further reduce the experienced noise and increase the number of supported end-devices per unit by squeezing each carrier signal in an ultra narrow-band (UNB) of width as short as 100Hz. However, the data rate decreases as well, thereby increasing the radio-on time. This combined with spectrum regulations on sharing underlying bands severely limits the maximum size and number of data packets. SigFox [3] is an example of a LPWAN technology that use UNB modulation.

2.1.2. Spread spectrum

Spread spectrum modulation techniques increase the link budget by spreading a narrow-band signal over a wider frequency band with the same power density. The resulting transmission is more resilient to interference, eavesdropping and jamming. However, decoding requires more processing gain and spreading results in lower spectrum efficiency. Different variants of spread spectrum techniques are used. LoRa [4] uses Chirp Spread Spectrum (CSS) while IEEE-802.15.4g [2], Weightless-P [5] and Ingenu [6] use Direct Sequence Spread Spectrum (DSSS).

2.1.3. OFDM based

The OFDM based modulation techniques used in IEEE-802.15.4g [2] and IEEE-802.11ah [7] sacrifice link budget for obtaining higher data rates. The transmission range is smaller, especially when considering the highest data rate. NB-IoT[8], standardised by 3GPP in LTE release 13, is another LPWAN tech-

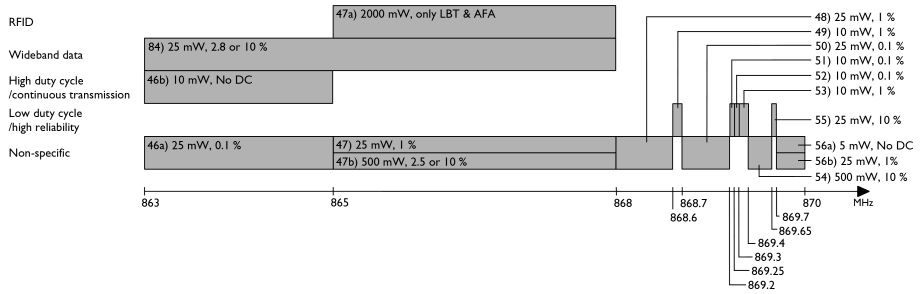


Figure 1: Frequency bands and their corresponding duty cycle and transmission power regulations in the EU 863-870 MHz range.

nology that uses OFDM. Because it operates in the licensed spectrum, it can achieve relatively high data-rates while still having a large coverage.

2.2. Sub-1 GHz ISM spectrum access.

To cope with the limited amount of available bandwidth in the sub-GHz unlicensed spectrum, duty cycle regulations are often enforced. For example, there are 6 sub-GHz frequency bands made available by EU law for non-specific short range devices in the 868MHz range and 2 in the 433MHz range. Each band has a specified maximum allowed transmission power and duty cycle ratio per device (see Figure 1). The maximum allowed power limits the maximum achievable transmission range. The duty cycle constraints impose a limit on the maximum amount of messages that the device can send each hour, thereby impacting the design of the routing and MAC protocol. Generally speaking, devices have to be conform to one or both of the following transmission constraints[9]:

- Duty cycle: devices in a frequency band are only permitted a maximum cumulative on air time per interval. EU law defines this interval as one hour. The duty cycles depend on the selected frequency band, but vary between 0.1% and 10% (see Figure 1).

- Polite spectrum access: devices implementing polite spectrum access are not bound by the hard duty cycle limit, but instead a maximum cumulative on air time of 100 s per hour for each possible 200 kHz interval is

imposed. Polite spectrum access is defined as the combination of Listen Before Talk (LBT) and Adaptive Frequency Agility (AFA). Devices must first check if the medium is free before transmission (LBT): if the medium is busy, then the device must wait or check another frequency (AFA).

These constraints combined with the low data rates strongly impact the feasibility for performing OTA updates.

2.3. Software update methods

Research into the application of software updates in LPWANs is very limited although it has been identified as one of the key research challenges for long term sustainability [10, 11, 12]. An overview of existing software update methods for constrained devices can be found in [13]. More details regarding different software update methods for LPWANs will be given in Section 4.

3. Energy consumption models for LPWANs

LPWAN devices need to operate multiple years on a single battery charge. As such, energy consumption is one of the prime criteria for evaluating the feasibility of LPWAN software updates. Surprisingly, experimental measurements that compare energy/power consumption of LPWAN devices could only be found in [14] which compares the performance (bit-rate vs. energy) for different LoRa modes with a proprietary narrow-band and ultra-narrow band solution. Therefore, in this section, different LPWAN technologies will be compared based on input gathered from radio transceiver data-sheets. The current selection is limited to radio modules that clearly specify the power consumption and receiver sensitivity information for a particular MCS configuration. The gathered data is summarised in Table 2. Each row lists the bit-rate (R), receiver sensitivity (RX_{sens}), transmitter output power (TX_{op}) and Rx/Tx power consumption for a specific MCS configuration of a technology (I_r/I_t). The last three columns contain the maximum transmission unit (MTU), PHY header size (PHS) and ack size (L_{ack}) as defined in the standard or technical specification.

Table 2: Bit-rate, receiver sensitivity, transmit output power and Rx/Tx power requirements for different technologies and MCS configurations.

Transceiver	Transceiver Configuration	R (kbit/s)	TX_{op} (dBm)	RX_{sens} (dBm)	I_r (A)	I_t (A)	MTU (bytes)	PHS (bytes)	L_{ack} (bytes)
SemTech SX1257	IEEE-802.15.4g MR-OFDM Option 3 16-QAM CR 3/4	600	5	-97	0.02	0.058	2047	12	7
Atmel AT86RF215	IEEE-802.15.4g MR-2FSK 50 ksymbols/s CS 200kHz	50	14	-114	0.015	0.056	2047	8	7
	IEEE-802.15.4g MR-OQPSK Chiprate 100kbit/s, Rate mode 0	6.25	14	-123	0.017	0.056	2047	10	7
SemTech SX1276	LoRa CSS SF7 CR 4/5 BW 250 kHz	10.938	14	-120	0.0116	0.044	250	-	5
	LoRa CSS SF7 CR 4/5 BW 125 kHz	5.4688	14	-123	0.0108	0.044	250	-	5
	LoRa CSS SF8 CR 4/5 BW 125 kHz	3.125	14	-126	0.0108	0.044	250	-	5
	LoRa CSS SF9 CR 4/5 BW 125 kHz	1.7578	14	-129	0.0108	0.044	123	-	5
	LoRa CSS SF10 CR 4/5 BW 125 kHz	0.9766	14	-132	0.0108	0.044	59	-	5
	LoRa CSS SF11 CR 4/5 BW 125 kHz	0.5371	14	-134.5	0.0108	0.044	59	-	5
	LoRa CSS SF12 CR 4/5 BW 125 kHz	0.293	14	-137	0.0108	0.044	59	-	5
Onsemi AXSF	SigFox UNB UL: DBPSK DL: GFSK	DL: 0.6 UL: 0.1	14	-126	0.01	0.049	DL: 8 UL: 12	DL: 20 UL: 14	0

3.1. Down-link range vs. energy consumption

Based on the input from Table 2 it is clear that increased down-link range (i.e. RX_{sens}) comes at the cost of a higher energy consumption and lower data rate. Figure 2 plots the charge (in micro Coulomb) required to receive one bit vs. the receiver sensitivity (in dBm).

- For IEEE-802.15.4g, the OFDM option has the lowest sensitivity, followed by the narrow-band option that increases the line-of-sight (LOS) range nearly by a factor 8 (e.g. +17dB) at the cost of a 9 time higher charge. The spread spectrum option (MR-QPSK DSSS) further increases the LOS range by a factor 3 (e.g. +9dB), again requiring 9 times more charge.

- For LoRa, increasing the spreading factor (SF) by one, always results in an increased LOS range of +-50% while requiring +- 1.8 time more electrical charge. A similar conclusion can be drawn when considering the bandwidth. Lowering the bandwidth by half, increases the LOS range by +-50% while using +- 1.8 time more electrical charge. The highest sensitivity is achieved by LoRa using SF12.

Figure 2 shows that the ultra narrow-band SigFox solution is somewhat an outlier compared to the other technologies. It achieves a high sensitivity of -126 but requires 4.8 time more charge compared to LoRa SF8 which has the same sensitivity. Moreover, SigFox UNB has a +- 50% longer range compared to the DSSS option of IEEE-802.15.4g but requiring 6 times more charge.

3.2. Link-layer energy model for down-link transactions

To estimate the energy costs of a software update (e.g. a down-link transaction), an appropriate energy model is required. Most existing energy models are tailored for a specific PHY/MAC combination[15, 16] making it hard to compare different technologies. For this reason, in this section a more generic analytic model considering only PHY layer bit-rate [17] will be created using the input data from Table 2.

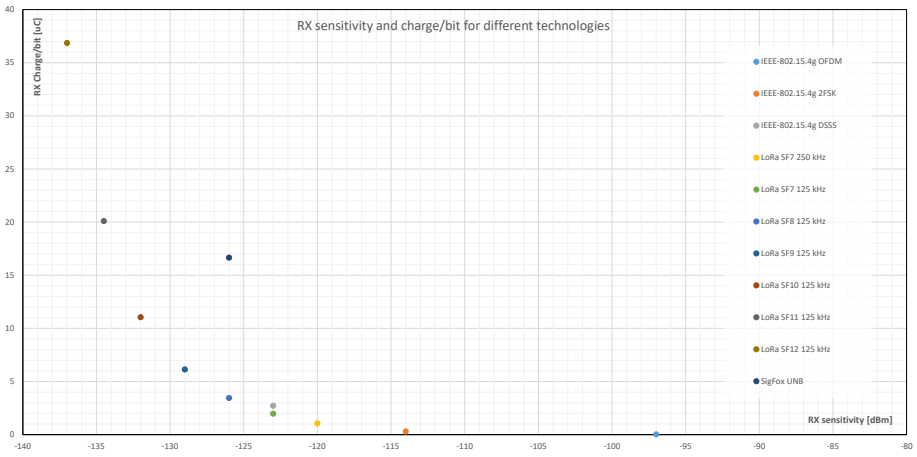


Figure 2: Energy consumption versus range trade-offs. X-axis: receiver sensitivity (in dBm) for different technologies and MCS configurations. Y-axis: electrical charge (in micro Coulomb) required to send one bit.

Equation 1 expresses an upper-bound for the energy required to receive/-transmit a single packet over a wireless link ($\overrightarrow{ij}$) including acknowledgement taking into account a certain success probability for the down- and up-link. The expression includes down- and up-link bit-rate (resp. R_d and R_u), message size in bits (resp. L_d and L_u) and success probability (resp. p_d and p_u). The power required to run the transmitter (receiver) circuitry is expressed by P_t (P_r). The formulae in Equation 1 have two parts: (1) send/receive a packet; (2) receive/send acknowledgement. The first part contains both the success probability for down- and uplink (resp. p_d and p_u) because a data packet retransmission can be triggered when loosing either a data packet or an acknowledgement. The second part (i.e. sending the ack) only occurs after a successful packet transmission/reception.

$$\begin{cases} E\left[e_t\left(i,j\right)\right] \leq P_t \frac{L_d}{p_d p_u R_d} + P_r \frac{L_u}{p_u R_u} \\ E\left[e_r\left(i,j\right)\right] \leq P_r \frac{L_d}{p_d p_u R_d} + P_t \frac{L_u}{p_u R_u} \end{cases} \tag{1}$$

Note that Equation 1 overestimates the energy because it assumes that all failed packets are fully received while in many cases the receiver can go to idle mode earlier.

3.3. Applying the model to different LPWAN technologies

In order to apply the aforementioned model on LPWANs, several assumptions are made.

- First, a single hop topology with a single mains-powered gateway is assumed. Therefore, only the energy consumption of the battery-powered end-devices is calculated (i.e. $E[e_r(i,j)]$).
- Second, it is also important to take into account the maximum payload size MTU, PHY header size PHS and ACK size (L_{ack}) because this has a non-negligible impact on the number of packets that need to be sent and the per packet PHY overhead (e.g. preamble, header, CRC, ..). These parameters are included in Code fragments 1 & 2.
- Third, sometimes bit-rate cannot be used directly to calculate the time-on-air (ToA). For instance, in LoRa the ToA is determined by the bandwidth (BW), spreading factor (SF) and coding rate (CR). For this reason, the generic calculation in Code fragment 1 is replaced by Code fragment 2 specific for LoRa.
- Finally, MAC overhead (i.e. headers and scheduling) is ignored. The motivation behind this is that MAC protocols are often highly configurable, allowing to fine-tune the behaviour for specific application requirements. Moreover, as LPWAN specifications are still work in progress, it can be expected that novel MAC protocols will be introduced, taking into account special application requirements such as OTA SW updates. In any case, energy models for specific MAC protocols can use the proposed PHY model as a basis.

Code fragments 1 and 2 demonstrate how the energy for a downlink transaction is calculated. First, *CalculateEnergyTotal* splits the total transaction in packets according to the MTU. Second, the ToA of both the packet and ack are calculated (resp. T_{packet} and T_{ack}), taking into account the bit-rate, PHS and ack length. This information is then passed to *CalculateEnergyPacket* that uses

Eq. 1 to calculate the energy cost. In code fragment 2, the ToA calculations are modified specifically for LoRa according to the formula defined in [18], using BW, SF and CR as input parameters. The function *CalculateEnergyTotal* has a time complexity of *O(n) with n the number of update messages that need to be sent.* The functions *CalculateEnergyPacket* and *CalculateEnergyLora* have O(1) time complexity.

```
def CalculateEnergyPacket(T_packet, T_ack, P_r, P_t, p_d, p_u):
    return (P_r * T_packet) / (p_d * p_u)) + ((P_t * T_ack) / p_u)

def CalculateEnergyTotal(L_total,R_d, R_u, p_d, p_u, P_r, P_t, PHS_d, PHS_u,
        MTU_d, MTU_u, L_ack):
    E_total = 0
    while L_total >= MTU_d:
        L_total -= MTU_d
        T_packet = (PHS_d + MTU_d) / R_d
        T_ack = (PHS_u + L_ack) / R_u
        E_total += CalculateEnergyPacket(T_packet, T_ack, P_r, P_t, p_d, p_u)
    if L_total > 0:
        T_packet = (PHS_d + L_total) / R_d
        T_ack = (L_ack + PHS_u) / R_u
        E_total += CalculateEnergyPacket(T_packet, T_ack, P_r, P_t, p_d, p_u)
    return E_total
```

Code fragment 1: Code fragment that calculates the energy consumption of an end-device for a down-link transaction

```
def CalculateEnergyLora(L_total, p_d, p_u, P_r, P_t, MTU_d, MTU_u, L_ack, SF
        , CR, BW):
    T_sym = 2^SF/BW
    T_preamble = (5 + 4.25) * T_sym
    E_total = 0
    while L_total >= MTU_d:
        L_total -= MTU_d
        N_sym_payload = 8 + Ceil((((MTU_d) - (4 * SF) + 28 + 16) / (4 * SF
            )), 1) * (CR + 4)
        T_packet = N_sym_payload * T_sym + T_preamble
        N_sym_ack = 8 + Ceil(((L_ack - 4 * SF + 28 + 16) / (4 * SF)), 1) *
            (CR + 4)
```

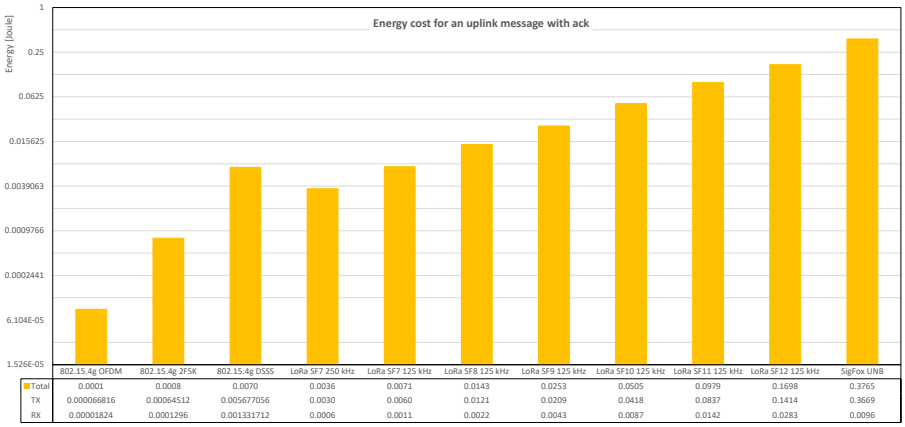


Figure 3: Energy (in Joule) required to send an up-link message containing 12 payload bytes and to receive an acknowledgement. A logarithmic scale base 2 is used.

```
    T_ack = N_sym_ack * T_sym + T_preamble
    E_total += CalculateEnergyPacket(T_packet, T_ack, P_r, P_t, p_d, p_u)
  if L_total > 0:
    N_sym_payload = 8 + Ceil((((L_total) - (4 * SF) + 28 + 16) / (4 * SF
        )), 1) * (CR + 4)
    T_packet = N_sym_payload * T_sym + T_preamble
    N_sym_ack = 8 + Ceil(((L_ack - 4 * SF + 28 + 16) / (4 * SF)), 1) *
        (CR + 4)
    T_ack = N_sym_ack * T_sym + T_preamble
    E_total += CalculateEnergyPacket(T_packet, T_ack, P_r, P_t, p_d, p_u)
  return E_total
```

Code fragment 2: Code fragment that calculates the energy consumption of an end-device for a down-link LoRa transaction

Using these formula's it is possible to calculate the energy cost for a particular down- or uplink transaction. For instance, Figure 3 depicts for each technology the energy cost in Joule to transmit 12 payload bytes and receive an acknowledgement (note that a logarithmic scale is used). The graph clearly shows that SigFox requires much more energy compared to the other technologies (i.e. 26.4 time more then the equal sensitive LoRa SF8 125 kHz).

4. Over-the-air update methods for LPWANs

Table 3 gives an overview of 3 different OTA software updates methods that can be applied in constrained LPWANs [13]:

- Firmware-based, with or without binary differential patching.
- Dynamic linking of binary code on the constrained devices, further divided based on the bindings between code blocks (either strict of loose).
- Pre-linking (i.e. offline on a more powerful computer) of binary code, again divided based on the binding type.

Table 3 also includes a qualitative analysis of performance indicators. The second column describes the scope to which code updates can be applied, varying from full firmware to application level only. The third and fourth column indicate respectively the bandwidth and latency requirements based on the number of bytes that need to be transferred. The fifth column sets forth the network disruption caused by the update before the normal network operation can resume. The last column estimates the end-to-end complexity of the particular update method.

The following subsections detail each of the aforementioned update methods.

4.1. Firmware-based

The least complex and most used approach for post-deployment code updates replaces the entire image. All source code is compiled into a single image and installed on each device. If an update is required, a new image must be compiled and distributed to all nodes. Because the entire image is re-programmable, the full scope of the firmware (OS, network and application) can be updated. Among all approaches, the bandwidth and latency overhead is the highest since the entire image must be distributed. Moreover, firmware updates require a system reboot and state recovery which is highly disruptive.

To make the update process more efficient in terms of bandwidth and latency, binary differential patching techniques can be used in order to reduce the size of

Table 3: Overview of software update methods for constrained LPWANs. The overview gives a qualitative analysis of performance indicators.

Update method		**Scope**	**Bandwidth**	**Latency**	**Disruptiveness**	**Complexity**
Firmware	Update	Full	High	High	High	Low
	Patch	Full	Medium-high	Medium-high	High	Medium
Dynamic linking	Strict binding	Applications	Low-medium	Low-medium	Low	Low-medium
	Loose binding	Network & Applications	Medium	Medium	Low-medium	Medium-high
Pre-linking	Strict binding	Applications	Low	Low	Low	Medium-high
	Loose binding	Network & Applications	Low-medium	Low-medium	Low-medium	High

the image that needs to be transferred. In this case, only the difference between the old and new binary (i.e. binary patch) is disseminated to each device where the new binary is reconstructed using the old binary and the patch file [19]. This comes however at the cost of an increased complexity. Patching techniques can also be applied to the methods discussed in the next subsections.

Currently there are a number of companies that are developing mechanisms to enable OTA firmware updates[20, 21, 22].

4.2. Dynamic linking

Another approach is to use a linker that is able to install software components at run-time in an active system. The linker relocates code (data) section to the allocated ROM (RAM) memory regions and resolves undefined references using a symbol look-up table. Since the individual components are smaller, the bandwidth and latency overhead is lower compared to firmware based solutions. The disruption is also lower because this approach does not require a system reboot and therefore state information can be transferred between updates. The complexity on the other hand increases because several task are required during installation (i.e. memory allocation, address relocation, undefined symbol look-up, etc.).

Solutions that rely on a dynamic linker can be further categorised by the binding model they use. The binding model defines how code blocks are linked post-deployment to the external functionality (functions, shared memory, ..) provided by code blocks already present. Currently, two models are applied: i) strict binding and ii) loosely coupled binding:

- The strict binding model, generally used in device firmwares, statically links code blocks to each-other, i.e. the linker replaces undefined symbols in one code block with the correct physical address of another code block. For this purpose, a run-time linker requires a global symbol table containing the memory addresses of all global symbols in the running firmware. Because the symbol table is generated before deployment, additions and

updates are restricted to component interactions that use pre-defined interaction functions.

- The loosely coupled binding model uses an indirect function call mechanism and jump tables to redirect function calls between code blocks. By manipulating the jump tables, it is now possible to update code blocks in the entire firmware, thereby extending the scope. On the other hand, the linking process is more complex as it requires to alter jump tables which could be embedded in each software component. Moreover, more information is required which increases the size of the update (and hence the bandwidth and latency).

4.3. Pre-linking with code injection

The task of the dynamic linker can also be offloaded to a more powerful server. Software components are now pre-linked before they are disseminated to the nodes. This strongly reduces the size of the update, requiring significantly less bandwidth and latency. On the other hand, pre-linking requires full knowledge about the firmware and memory map of each device in order to execute the same task as the dynamic linker. Moreover, the initial firmware must also be adapted in order to support code injection. This makes it more complex, especially in heterogeneous networks.

Again, there are two options based on the required scope. If only top level applications need to be added/updated, a strict binding model can be applied and the code block can simply be injected, only requiring a starting point to activate the component. Otherwise, a loosely coupled binding model should be applied. This implies that the jump tables are modified, requiring more information and a complex loading process.

4.4. Script interpreters

Another possible method, not mentioned in Table 3, relies on script interpreters for enabling upgrade-ability. Due to the run-time interpretation, scripts

can be added or updated after deployment. Some well-known scripting languages like Python[23] and JavaScript[24] were already ported to embedded devices. Despite this, they still require a substantial amount of memory and CPU overhead. Moreover, they work on top of existing operating system and network stack functionality, limiting the scope to the application layer. For these reasons scripts are not suited for the low-capability hardware platforms targeted in this paper.

5. Feasibility of OTA software updates in LPWANs

In order to evaluate the feasibility of OTA software updates in LPWANs, the size of the update must be known. First, the transaction size is determined, i.e. the minimal number of bytes that need to be transferred to a device in order to allow a software update using a particular update method. Afterwards, the energy cost is estimated for the different LPWAN technologies based on the energy models discussed in Section 3. All results presented in this paper and an implementation of the energy models, can be accessed online [25].

Three different scenario's are considered:

- Full operating system update. For example, updating the embedded OS from Contiki 3.0 to Contiki 3.1.
- Single application update. For example, upgrading a simple application to a more robust applications with acknowledgements and retransmissions.
- Low level network protocol update. For example, updating the MAC protocol from ContikiMAC 3.0 to ContikiMAC 3.1.

In the first scenario only a full firmware update is possible, since the entire code is susceptible to changes in a version update. In the second scenario all update methods described in Section 4 are possible, while in the third scenario, only firmware-based and code compiled with a loosely coupled binding model can be

used[1].

Table 4 lists the update sizes of the new executable (i.e. down-link transaction size) and the installation energy cost for each of the investigated scenario's and methods. The size is determined by analysing the size of the executable object files (i.e. ELF Executable and Linkable Format[26]) generated when building Contiki operating system in the different update scenarios. At a first glance, it also seams surprising that an application update requires more bytes than a version update. This is because the application update scenario was created in Contiki 3.1 and logic (i.e. ROM/RAM) was added, hence the larger file size. The overall impact on the energy consumption will be investigated in the following subsections.

Table 4 also estimates the minimal installation energy cost, calculated using Equation 2. It is defined as the cost to write n_{elf} bytes to store the ELF file, read the same n_{elf} bytes for processing (linking and relocating) and copying the processed ROM and RAM bytes (resp. n_{rom} and n_{ram}) to the correct memory location. The energy cost to write/read a single byte to ROM/RAM is denoted as E^{write}_{rom}, E^{read}_{rom}, E^{write}_{ram} and E^{read}_{ram} and listed in Table 5 for the 32-MhZ CC2538 micro-controller[2]. Note that the CPU processing for the ELF file is not accounted, only the copy and read operations are calculated.

$$E_{install}\left(n_{elf}, n_{rom}, n_{ram}\right) > n_{elf} \times E^{write}_{rom} + n_{elf} \times E^{read}_{rom} + n_{rom} \times E^{write}_{rom} + n_{ram} \times E^{write}_{ram} \quad (2)$$

The terms in Equation 2 estimates the cost to: (1) write the elf file to ROM; (2) read the ELF file for processing; (3) write the relocated ROM; and (4) write the relocated RAM.

[1] Code blocks that use a strict binding model only allow partial updates of top-level applications.

[2] http://www.ti.com/lit/ds/symlink/cc2538.pdf [last accessed on 17/07/2018]

Table 4: Update sizes in three different scenarios (full system update, application update and MAC update) for different software update methods. The energy cost during installation is also given.

Scenario	Method	Update size (bytes)	Install Energy Cost (Joule)
Full System Update	Firmware-based	23876	0.017897594
App update	Firmware-based	24632	0.018470381
	Dyn. linking strict binding	2000	0.000880547
	Dyn. linking loose binding	3824	0.001785231
	Pre-linking strict binding	716	0.000387684
	Pre-linking loose binding	1304	0.000817929
MAC update	Firmware-based	24012	0.018001584
	Dyn. linking loose binding	7016	0.003618
	Pre-linking strict binding	2908	0.002041144

E_{rom}^{write}	0.000000378
E_{rom}^{read}	5.85E-09
E_{ram}^{write}	5.85E-09
E_{ram}^{read}	5.85E-09

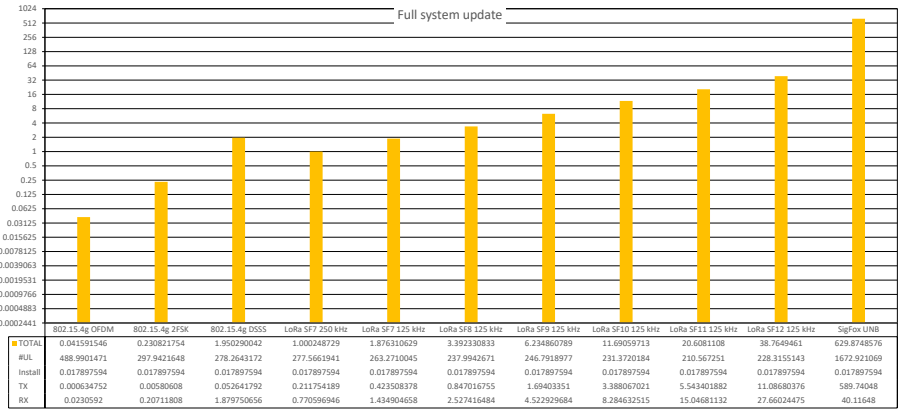
Table 5: Input parameters for calculating the installation energy cost on a 32-MHz CC2538 micro-controller.

5.1. Scenario 1: full system update

A full firmware upgrade is the only viable option for performing version updates of operating systems (e.g. from Contiki 3.0 to Contiki 3.1) because such an update affects a large part of the code base. Dividing a version update in multiple smaller updates, i.e. only the changed ELF object files, will have a higher overall ELF overhead (i.e. an ELF file contains several headers, allowing to relocate, link and load the binary code). In any case, a version update will have the highest transactions size and, consequently, energy cost compared to the other scenario's.

Figure 4 depicts the energy cost in Joule for the different technologies. All calculations assume a battery powered system using a single Lithium-Thionyl-Chloride (LTC) AA battery with an average output voltage of 3.6V and an energy capacity of 2.4Ah (31104 Joule). The chart shows the overall energy consumed during the update. Note that a logarithmic scale base 2 is used. The datatable below includes the actual numbers for transmit (TX), receive (RX) and installation (Install), as well as the number of 12-byte up-link messages (#UL) including acknowledgements that could have been sent with the same energy budget.

The data shows that all three IEEE-802.15.4g options have relatively low TX overheads. Due to the large IEEE-802.15.4g MTU sizes (i.e. 2047 bytes), the total number of received packets is small, resulting in less uplink acknowledgements and hence limited TX overhead. In contrast, due to the MTU restrictions of SigFox (i.e. max 8 bytes), much more acks are sent and consequently more



	802.15.4g OFDM	802.15.4g 2FSK	802.15.4g DSSS	LoRa SF7 250 kHz	LoRa SF7 125 kHz	LoRa SF8 125 kHz	LoRa SF9 125 kHz	LoRa SF10 125 kHz	LoRa SF11 125 kHz	LoRa SF12 125 kHz	SigFox UNB
TOTAL	0.041591546	0.230821754	1.950290042	1.000248729	1.876310629	3.392330833	6.234860789	11.69059713	20.6081108	38.7649461	629.8748576
#UL	488.9901471	297.9421648	278.2643172	277.5661941	263.2710045	237.9942671	246.7918977	231.3720184	210.567251	228.3155143	1672.921069
Install	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594	0.017897594
TX	0.000634752	0.00580608	0.052641792	0.211754189	0.423508378	0.847016755	1.69403351	3.388067021	5.543401882	11.08680376	589.74048
RX	0.0230592	0.20711808	1.879750656	0.770596946	1.434904658	2.527416484	4.522929684	8.284632515	15.04681132	27.66024475	40.11648

Figure 4: Energy cost (in Joule) for performing a full system update for different LPWAN technologies. Note that a logarithmic scale base 2 is used.

energy is spent in TX. This and the ultra low data-rate are the main reason why SigFox requires more energy overall, even compared to LoRa SF12 (e.g. a factor 16 more) which has a higher sensitivity. Another interesting observation is the constant rate in which the energy cost increases for the different LoRa options. For a three dBm increase in RX sensitivity, 1.8 times more energy is required.

Figure 5 illustrates the distribution of the energy cost in percentage between RX, TX and Install. In most cases, RX contributes the most to the overall energy usage except for SigFox as explained earlier. In contrast, TX energy consumption is relatively low for the IEEE-802.15.4g technologies due to the higher MTU. Also notably is the low contribution of the installation in the overall energy cost, except for IEEE-802.15.4g-OFDM. Generally, the higher the data rate, the higher the installation overhead.

Note that all aforementioned observation holds for all subsequent scenario's, i.e. the trend in the differences between technologies is similar. The subsequent scenario's will hence focus on the differences between update methods for a number of technologies.

When considering the overall energy capacity of the LTC battery, all technologies consume less than 0.125% except SigFox which requires 2%. It is hence

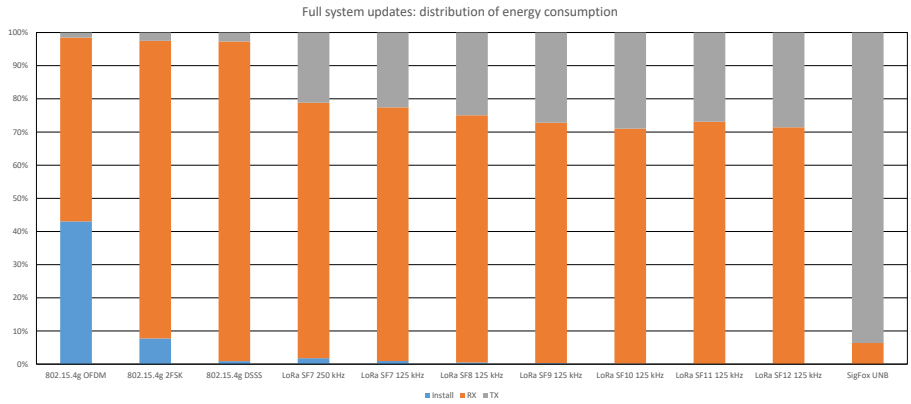


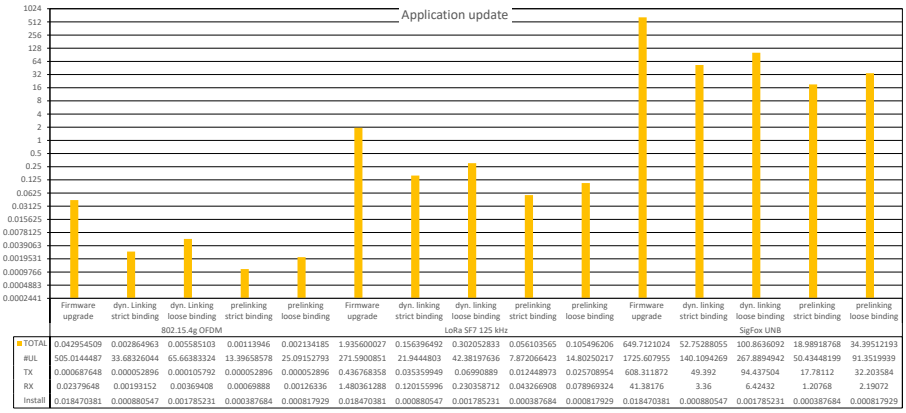
Figure 5: Distribution of the energy cost in percentage between RX, TX and Install for the different LPWAN technologies.

arguable that software updates are feasible but this also depends on the overall requirements in terms of battery lifetime and application data rate. To put this in perspective, for SigFox a version update equals 1672 up-link messages in terms of energy cost. This is actually quite a high number (e.g. equals 69 days of energy for an hourly report interval). A version update for LoRa SF12 only costs the same energy as 228 up-link messages, reducing the battery lifetime 9.5 days, which is much more reasonable.

5.2. Scenario 2: application updates

For application updates all methods discussed in Section 4 can be used. In this case, a simple application that reports sensor data upstream was extended with acknowledgements and re-transmissions. As all methods can be applied, it is a good case study to compare the update methods. Figure 6 depicts the energy cost for three different LPWAN technologies (IEEE-802.15.4g-OFDM, LoRa BW 125 kHz SF7 and SigFox UNB) using a firmware upgrade , a dynamic linker with strict or loose binding model and a pre-linker again with strict or loose binding model.

When comparing the different update methods, firmware updates will have a higher energy cost compared to the alternatives because the size of the update



	802.15.4g OFDM					LoRa SF7 125 kHz					SigFox UNB				
TOTAL	0.042954509	0.002864963	0.005585103	0.00113946	0.002134185	1.935600027	0.156396492	0.302052833	0.056103565	0.105496206	649.7121024	52.75288055	100.8636092	18.98918768	34.39512193
#UL	505.0144487	33.68326044	65.66383324	13.39658578	25.09152793	271.5900851	21.9444803	42.38197636	7.872066423	14.80250217	1725.607955	140.1094269	267.8894942	50.43448199	91.3519939
TX	0.000687648	0.000052896	0.000105792	0.000052896	0.000052896	0.436768358	0.035359949	0.06990889	0.012448973	0.025708954	608.311872	49.392	94.437504	17.78112	32.203584
RX	0.02379648	0.00193152	0.00369408	0.00069888	0.00126336	1.480361288	0.120155996	0.230358712	0.043266908	0.078969324	41.38176	3.36	6.42432	1.20768	2.19072
Install	0.018470381	0.000880547	0.001785231	0.000387684	0.000817929	0.018470381	0.000880547	0.001785231	0.000387684	0.000817929	0.018470381	0.000880547	0.001785231	0.000387684	0.000817929

Figure 6: Depicts the estimated energy cost in Joule for performing an application update for three different LPWAN technologies (IEEE-802.15.4g-OFDM, LoRa BW 125 kHz SF7 and SigFox UNB) using the different update methods. The datatable lists the energy usage in each stage (RX, TX, Install) and the number of 12 byte UL messages (including acknowledgement) that could be sent with the same energy budget. The graphs use a logarithmic scale base 2.

is much bigger. Compared to the other methods, firmware upgrades require respectively 12.3 (dyn. linking strict binding), 6.4 (dyn. linking loose binding), 34.2 (pre-linking strict binding) and 18.5 times (pre-linking loose binding) times more energy on average.

- Using a pre-linker reduces the energy cost with a factor 2.8 compared to a dynamic linker. This reduction is caused entirely by the smaller file size. A linked ELF file only contains the actual code, ELF file header and program headers. All relocation entries, the symbol table and string table can be omitted from the file. On the other hand, this method requires that the pre-linker is perfectly aware of the existing firmware memory map and the free memory locations of each device.

- Using a loose binding model (i.e. code that can be re-linked at run-time) increases the energy cost with a factor 1.9 compared to a strict binding model. The larger file size is a consequence of the increase in ROM/RAM required to make the code re-linkable at run-time by replacing direct function calls with indirect calls using function pointers in a jump table.

- To update or add an application with a pre-linked ELF file that uses strict binding only requires less then 10 uplink messages for most technologies except SigFox that requires 50 uplink messages. The dynamic variant requires less then 0.007% more battery for most technologies (0.10% for SigFox). Using a loose binding model with a dynamic linker requires between 34 and 267 uplink messages. Combining a loose binding model with a pre-linker reduces this further to 12 and 91 uplink messages.

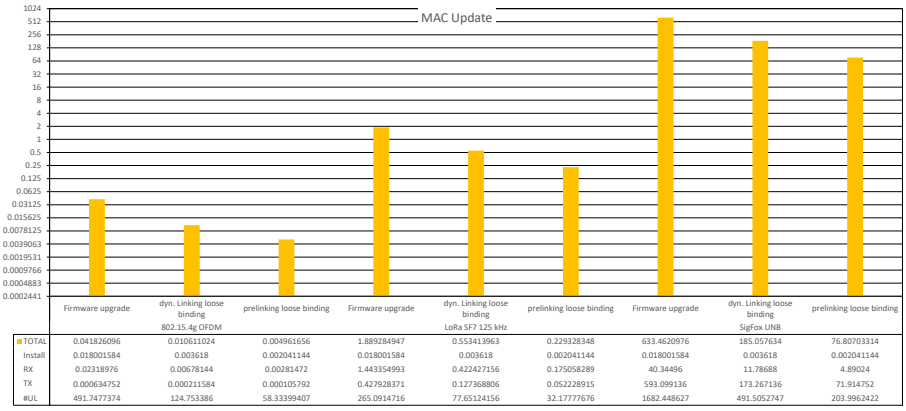
Overall, it can be concluded that adding or updating an application is feasible using all methods. By applying more sophisticated methods, instead of firmware updates, the overall energy cost can be greatly reduced. Among these alternatives there is a clear trade-off between energy cost and update scope, i.e. allowing more modules to be updated requires more energy. Moreover, there is also a trade-off between energy cost and update complexity, i.e. a more complex update system using a pre-linker consumes less energy on the end-device. For the specific application update considered in this example, over-the-air updates require between 84% and 97% less energy than a full firmware update, depending on the update method.

5.3. Scenario 3: MAC updates

The third scenario considers the upgrade of a MAC protocol because MAC protocols have a large impact on the energy consumption as they directly control the radio, which dominates the overall energy usage. Moreover, given the complexity of the algorithms inside MAC protocols they are also very susceptible to software bugs. This makes them a primary targets for software updates.

Due to their strong interaction with hardware components and other network protocols, networking protocols can only be updated using firmware updates or by using a loose binding model that allows re-linking code blocks at run-time. For the latter, again two linker options are available: a) a dynamic linker, i.e. on the device; b) a pre-linker, on the gateway/update server.

Figure 7 depicts the energy cost in Joule using each of these methods for three different LPWAN technologies (IEEE-802.15.4g-OFDM, LoRa BW 125



	Firmware upgrade	dyn. Linking loose binding 802.15.4g OFDM	prelinking loose binding	Firmware upgrade	dyn. Linking loose binding LoRa SF7 125 kHz	prelinking loose binding	Firmware upgrade	dyn. Linking loose binding SigFox UNB	prelinking loose binding
TOTAL	0.041826096	0.010611024	0.004961656	1.889284947	0.553413963	0.229328348	633.4620976	185.057634	76.80703314
Install	0.018001584	0.003618	0.002041144	0.018001584	0.003618	0.002041144	0.018001584	0.003618	0.002041144
RX	0.02318976	0.00678144	0.00281472	1.443354993	0.422427156	0.175058289	40.34496	11.78688	4.89024
TX	0.000634752	0.000211584	0.000105792	0.427928371	0.127368806	0.052228915	593.099136	173.267136	71.914752
#UL	491.7477374	124.753386	58.33399407	265.0914716	77.65124156	32.17777676	1682.448627	491.5052747	203.9962422

Figure 7: Depicts the estimated energy cost in Joule for performing a MAC update for three different LPWAN technologies (IEEE-802.15.4g-OFDM, LoRa BW 125 kHz SF7 and SigFox UNB) using update methods that allow MAC updates (firmware updates or dynamic/pre-linking with loose binding). The datatable lists the energy usage in each stage (RX, TX, Install) and the number of 12 byte UL messages (including acknowledgement) that could be sent with the same energy budget. The graphs use a logarithmic scale base 2.

kHz SF7 and SigFox UNB).

- The overall energy cost of a firmware-based MAC update is similar to the previous two scenarios.
- Updating a MAC protocol using a dynamic linker reduces the number of uplink messages that can be sent between 62 (LoRa SF11, 125 kHz) and 491 (SigFox), while decreasing the energy cost by a factor 3.5 compared to a full firmware update.
- Using a pre-linker decreases the average energy cost with a factor 8.2 compared to a full firmware update and 2.3 compared to a dynamic linker.

From these results it can be concluded that MAC updates are feasible in LPWANs, especially when using software update methods with a loose binding model.

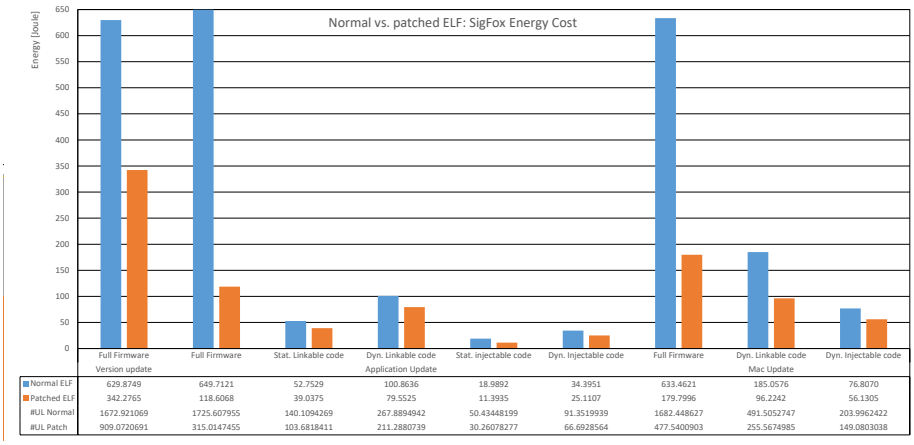


Figure 8: Impact of binary differential patching techniques for the different update scenario's for SigFox. Note that a linear scale is used.

5.4. Impact of binary differential patching

The results shown until now did not use binary differential patching techniques for reducing the number of bytes that need to be transferred. However, despite the additional complexity and memory usage, it is worth investigating how much energy can be saved using differential patching mechanisms.

Figure 8 illustrates the energy savings that can be made for SigFox when applying patching techniques in the different update scenario's. Note that a linear scale is used.

$$
\begin{aligned}
&E_{install}^{patch}\left(n_{patch}, n_{elf}^{old}, n_{elf}^{new}, n_{rom}, n_{ram}\right) > \\
&\quad n_{patch} \times E_{rom}^{write} + n_{patch} \times E_{rom}^{read} + n_{elf}^{old} \times E_{rom}^{read} \\
&\quad + n_{elf}^{new} \times E_{rom}^{write} + n_{elf}^{new} \times E_{rom}^{read} + n_{rom} \times E_{rom}^{write} + n_{ram} \times E_{ram}^{write}
\end{aligned}
\tag{3}
$$

The size of the patched update listed in Table 6 were obtained using JojoDiff[27], a patch utility tailored for memory constrained devices. Compared to Table 4, the patched version clearly lowers the number of bytes that need to be transferred but increases the energy required during installation because the new file must be created based on the old file and the patch. The first three terms in

Table 6: Patched update sizes in three different scenarios (full system update, application update and MAC update) for different software update methods. The energy cost during installation is also given.

Scenario	Method	Patch size (bytes)	Install Energy Cost (Joule)
Full System Update	Firmware-based	12973	0.023006893
App update	Firmware-based	4496	0.020335845
	Dyn. linking strict binding	1479	0.001457341
	Dyn. linking loose binding	3015	0.00295927
	Pre-linking strict binding	429	0.000555842
	Pre-linking loose binding	951	0.00118875
MAC update	Firmware-based	6816	0.020758492
	Dyn. linking loose binding	3647	0.005061987
	Pre-linking strict binding	2127	0.00287533

Equation 3 reflect the additional operations compared to Equation 2 when using differential patching changes: (1) write n_{patch} bytes to store the patch; (2) read n_{patch}; and (3) n_{elf}^{old} bytes to create the new ELF file based on the patch and previous ELF file. The energy cost to write/read a single byte to ROM/RAM is denoted as E_{rom}^{write}, E_{rom}^{read}, E_{ram}^{write} and E_{ram}^{read} and listed in Table 5.

Non surprisingly, the biggest reduction is obtained when applying patching techniques on full firmware update method lowering the energy cost with a factor 1.8 (version update), 5.5 (application update) and 3.5 (mac update). Nevertheless there is still a significant difference with the other methods especially when considering the reduction in uplink messages that can be sent after the update. The impact of patching on the other methods is less prominent except for the dynamic linker of a MAC update using code with a loose binding model (i.e. a factor 1.9).

5.5. Impact of packet loss and link symmetricity

The previous results assume symmetric links and a packet delivery ratio of 100% both in the downlink and uplink (i.e. $p_d = p_u = 1$ in Equation 1). In

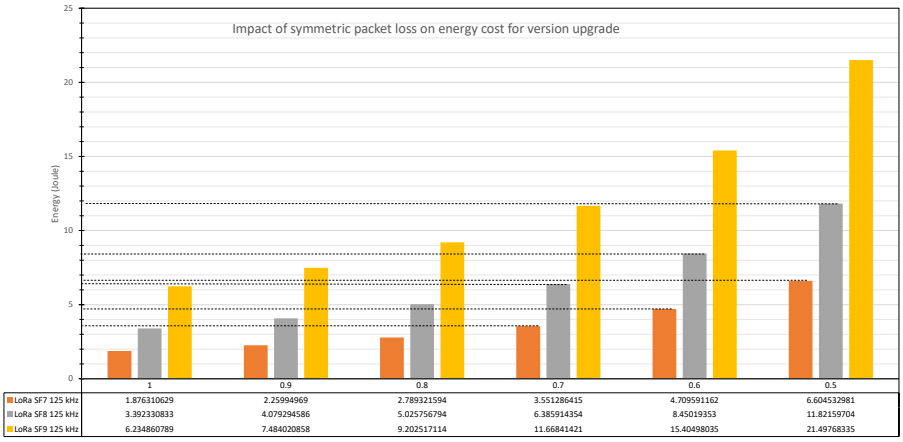


Figure 9: Impact of symmetric packet loss on the energy cost for performing a firmware based full system update for 3 different LoRa spreading factors. The striped horizontal lines indicate under which packet loss conditions it is better to switch to another spreading factor. Note that a linear scale is used.

realistic scenario's however, this will not be the case.

Figure 9 illustrates the effect of symmetric packet loss on the energy cost for performing a firmware-based version update. Note that a linear scale is used. Three LoRa spreading factors are considered (SF 7, 8 & 9). On average, the energy cost increases with a factor 1.28 if the PDR drops with 10%. When a lot of packet loss occurs, a lower modulation rate can be used. The striped horizontal lines indicate under which packet loss conditions it is better to switch. For instance, when 50% of the packets are lost using LoRa SF 7, the transceiver should switch to LoRa SF 8 if the same link only suffers from 30% packet loss with the new modulation setting.

Figure 10 illustrates the effect of asymmetric packet loss on the energy cost for performing a firmware-based version update. The results show a decreasing PDR in the uplink (from 100% to 50%) while the downlink PDR remains at 100%. Again, three LoRa spreading factors are considered (SF 7, 8 & 9) and a linear scale is used on the Y-axis. On average, the energy cost increases with a factor 1.15 if the PDR drops with 10%. This is +- 15% less compared to symmetric packet loss. It is however less interesting to downscale the modulation rate. The transceiver should only switch when 50% of the packets are lost using LoRa SF 7 (SF 8) and the same link using LoRa SF 8 (SF 9) has no packet loss.

6. Conclusion

This paper investigates the feasibility of providing over-the-air software updates for emerging LPWAN technologies. For this purpose, the down and uplink energy usage of several LPWAN technologies (i.e. IEEE-802.15.4g, LoRa and SigFox) was compared for three different scenarios: a full system update, application updates and network protocol updates. Table 7 summarises the results and indicates the most efficient update method in for each scenario. Ideally, full system updates are done using firmware updates with differential patches, application updates are done using pre-linked code with strict binding models

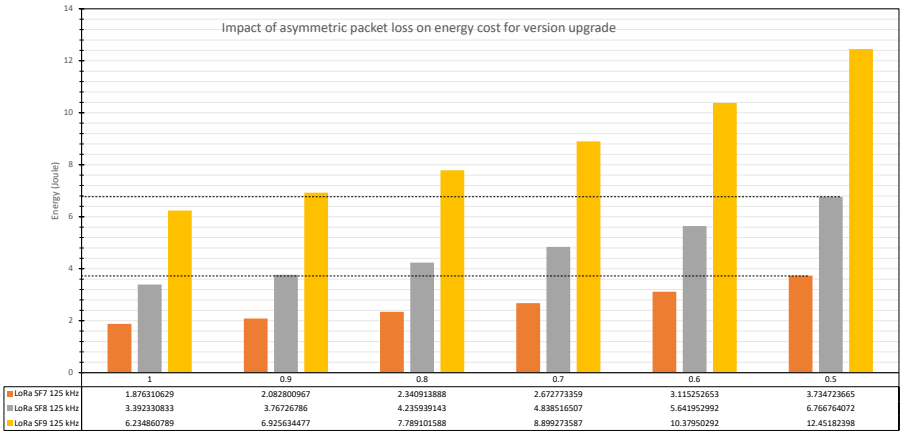


Figure 10: Impact of asymmetric packet loss on the energy cost for performing a firmware based full system update for 3 different LoRa spreading factors. The striped horizontal lines indicate under which packet loss conditions it is better to switch to another spreading factor. Note that a linear scale is used.

and network stack updates using pre-linked code with loose binding models.

The results indicate that software updates, and downlink transactions in general, are feasible in LPWANs. Nevertheless, full firmware upgrades consume a substantial amount of energy, especially for the lowest bit-rate LPWAN technologies such as SigFox which drains the batteries with 1.1% when performing a version update, compared to only 0.000115% for the OFDM based IEEE-802.15.4g technology. The results also show that 10% packet loss leads to +-30% increase in energy usage for symmetric links and +-15% increase for asymmetric links (i.e. no packet loss in downlink). In the former case it is better to switch to a lower bitrate modulation, if the packet loss is +-20% less using the lower bitrate modulation.

In contrast, application updates and network protocol updates require between 0.000004% and 0.25% of the battery depending on the update method and LPWAN technology.

Overall, it can be concluded that over-the-air updates are possible even for constrained LPWAN networks on condition that a suitable update approach is selected. As such, over-the-air updates will become increasingly important in

Table 7: Overview of the most efficient update method for the three scenarios. For each scenario, the method is displayed that uses least amount of energy.

Scenario	Update method
Version Update	Firmware update using patches
App. Update	Pre-linking strict binding model
MAC Update	Pre-linking loose binding model

LPWANs to cope with changing network requirements and increased security needs.

Acknowledgment

Part of this research was funded by the Flemish FWO SBO S004017N IDEAL-IoT (Intelligent DEnse And Longe range IoT networks) project, the MuSCLe-IoT (Multimodal Sub-Gigahertz Communication and Localization for Low-power IoT applications) project, co-funded by imec, a research institute founded by the Flemish Government, with project support from VLAIO (contract number HBC.2016.0660). ***

References

References

[1] U. Raza, P. Kulkarni, M. Sooriyabandara, Low power wide area networks: An overview, IEEE Communications Surveys Tutorials 19 (2) (2017) 855–873. doi:10.1109/COMST.2017.2652320.

[2] IEEE, Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans) amendment 3: Physical layer (phy) specifications for low-data-rate, wireless, smart metering utility networks, IEEE Std 802.15.4g-2012 (Amendment to IEEE Std 802.15.4-2011) (2012) 1–252doi:10.1109/IEEESTD.2012.6190698.

[3] SigFox. Sigfox specification [online] (2017).

[4] L. Alliance. Lora specification [online] (2017).

[5] W. SIG. Weightless specification [online] (2017).

[6] Ingenu. Ingenu specification [online] (2017).

[7] IEEE, Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 2: Sub 1 ghz license exempt operation, IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016, as amended by IEEE Std 802.11ai-2016) (2017) 1–594doi:10.1109/IEEESTD.2017.7920364.

[8] D. Flore, 3gpp standards for the internet-of-things (2016).

[9] E. T. S. Institute, Short range devices (srd) operating in the frequency range 25 mhz to 1 000 mhz; part 1: Technical characteristics and methods of measurement, ETSI Standard EN 300 220-1, ETSI, version 3.1.1 (02 2017).

[10] P. Thubert, A. Pelov, S. Krishnan, Low-power wide-area networks at the ietf, IEEE Communications Standards Magazine 1 (1) (2017) 76–79. doi:10.1109/MCOMSTD.2017.1600002ST.

[11] E. De Poorter, J. Hoebeke, M. Strobbe, I. Moerman, S. Latré, M. Weyn, B. Lannoo, J. Famaey, Sub-ghz lpwan network coexistence, management and virtualization: An overview and open research challenges, Wireless Personal Communications 95 (1) (2017) 187–213. doi:10.1007/s11277-017-4419-5.

[12] B. Moran, M. Meriac, H. Tschofenig, D. Brown, A Firmware Update Architecture for Internet of Things Devices, Internet-Draft draft-ietf-suit-architecture-01, Internet Engineering Task Force, work in Progress (Jul. 2018).

[13] P. Ruckebusch, E. D. Poorter, C. Fortuna, I. Moerman, Gitar: Generic extension for internet-of-things architectures enabling dynamic updates of network and application modules, Ad Hoc Networks 36 (2016) 127 – 151. doi:https://doi.org/10.1016/j.adhoc.2015.05.017.

[14] S. Kartakis, B. D. Choudhary, A. D. Gluhak, L. Lambrinos, J. A. McCann, Demystifying low-power wide-area communications for city iot applications, in: Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization, WiNTECH '16, ACM, New York, NY, USA, 2016, pp. 2–8. doi:10.1145/2980159.2980162.

[15] B. Martinez, M. Montn, I. Vilajosana, J. D. Prades, The power of models: Modeling power consumption for iot devices, IEEE Sensors Journal 15 (10) (2015) 5777–5789. doi:10.1109/JSEN.2015.2445094.

[16] A. Bel, T. Adame, B. Bellalta, An energy consumption model for IEEE 802.11ah wlans, CoRR abs/1512.03576. arXiv:1512.03576.
URL http://arxiv.org/abs/1512.03576

[17] J. Vazifehdan, R. V. Prasad, M. Jacobsson, I. Niemegeers, An analytical energy consumption model for packet transfer over wireless links, IEEE Communications Letters 16 (1) (2012) 30–33. doi:10.1109/LCOMM.2011.111611.110729.

[18] SemTech. Sx1272/3/6/7/8: Lora modem. designers guide an1200.13 [online] (2017).

[19] M. Stolikj, P. J. L. Cuijpers, J. J. Lukkien, Efficient reprogramming of wireless sensor networks using incremental updates, in: 2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013, pp. 584–589. doi:10.1109/PerComW.2013.6529563.

[20] Tracknet. Fota [online] (2018).

[21] Everynet. Fota. [online] (2018).

[22] LinkLabs. Fota. [online] (2018).

[23] Micropython. Python for microcontrollers [online] (2018).

[24] E. Baccelli, J. Doerr, S. Kikuchi, F. A. Padilla, K. Schleiser, I. Thomas, Scripting Over-The-Air: Towards Containers on Low-end Devices in the Internet of Things, in: IEEE PerCom 2018, Athens, Greece, 2018.

[25] P. Ruckebusch, E. De Poorter, J. Hoebeke, S. Giannoulis, I. Moerman, Energy consumption model for over-the-air software updates in lpwan networks: Sigfox, lora and ieee 802.15.4g., Mendeley Data, http://dx.doi.org/10.17632/ksmb3hxckj.1 (08 2018).

[26] Executable and linkable format (elf), tool Interface Standards Committee and others (2001).

[27] J. Heirbaut. Jojodiff: diff utility for binary files. [online] (2018).