LPWA通信を利用するIoTデバイス向け軽量スクリプトの設計と実装

松山凌空[1,a)]

概要：LPWA 通信を利用する IoT 端末は低速・狭帯域であるため，OTA によるファームウェア更新は通信負荷が大きく，運用中の機能変更が困難である．本研究では，ベースファームウェアを固定し，更新対象をスクリプトに限定して，端末上の独自 VM で実行する方式を提案・実装した．温度計測の例ではデータ量は 54B となり，既存の軽量言語処理系である mruby/c（215B）や Lua（621B）と比較して更新データ量を削減しつつ，平均 3.15 秒での遠隔更新に成功した．また，スタックマシン型の最小 VM 設計により，実行時 RAM 使用量 0.68KB を実現した．本方式により，低リソースな LPWA モジュールにおける柔軟な運用の実現可能性を示した．

キーワード：LPWA，IoT，仮想マシン（VM），スクリプト言語，遠隔更新

Design and Implementation of a Lightweight Scripting Language for IoT Devices Using LPWA Communication

Matsuyama Riku[1,a)]

Abstract: IoT devices using LPWA networks are hard to update in the field because conventional over-the-air firmware updates impose high communication overhead in low-data-rate, narrow-bandwidth environments. This study proposes and implements an approach that keeps the base firmware fixed and restricts update targets to scripts executed on a custom VM running on the device. In a temperature-measurement scenario, the update payload was reduced to 54B, compared with 215B for mruby/c and 621B for Lua, while achieving remote updates in 3.15s on average. Furthermore, a minimal stack-based VM design reduced runtime RAM usage to 0.68KB. These results demonstrate the feasibility of flexible in-field operation and maintenance for resource-constrained LPWA modules.

Keywords: LPWA, IoT, virtual machine（VM）, scripting language, remote update

1. 研究背景

近年，農業や防災をはじめとする分野で IoT の導入が進み，遠隔地に設置された多数の端末から計測データを収集・制御する需要が増大している．これらの端末は屋外や広域に分散して配置されることが多く，電池交換や保守作業のための現地作業が運用コストの増大要因となる．そのため，端末側には省電力で長期運用可能であることが求められ，広域通信を低消費電力で実現できる通信技術として LPWA（Low Power Wide Area）が注目されている．一方で，長期運用を前提とする IoT システムでは，不具合修正や機能追加などの観点からも運用中のソフトウェア更新が不可欠であり，現地作業を伴わず通信経由で更新可能な遠隔更新は運用負荷の低減に有効である．

遠隔更新の代表的な手法として OTA（Over-the-Air）更新が広く用いられている．OTA は，通信経由で機器のソフトウェアを更新する方式で，現地作業を伴わずに機能追加や不具合修正を行える点が利点である．しかし，LPWA

[1] 九州工業大学
Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[a)] matsuyama.riku649@mail.kyutech.jp

は低速・狭帯域であるため，ファームウェア全体を書き換える OTA では更新データ量が大きく，更新完了までに長時間を要するという課題がある．例えば，LoRaWAN を用いた先行研究においては，約 100KB のファームウェアを転送するために 100 分以上の時間を要することが報告されている [1]．これは LPWA の通信速度が低いことやパケット分割の多さが要因であり，多数の端末を運用する環境において，この転送時間は実用上の大きな障壁となる．

この課題に対し，組込みスクリプト言語の活用が有効なアプローチとなる．組込みスクリプト言語は，動作ロジックのみをスクリプトで記述できるため，ファームウェア全体を更新する場合と比較して更新データ量は大幅に削減できる．しかし，代表的な軽量スクリプト言語である mruby/c や Lua であっても，実行にメモリ使用量が 20KB 以上のメモリを必要とする [2][3]．多くの LPWA 通信モジュールは省電力・低コスト設計の制約から搭載メモリが小さく，これらの軽量スクリプト言語をそのまま実装することは困難である．例えば，総 RAM 容量が 64KB のモジュールにおいて，通信スタック等の基本機能が大部分を占有し，スクリプト実行に割り当て可能な領域は残り十数 KB 程度に制限される．この制約下では，実行に 20KB 以上を要する既存の軽量スクリプト言語の導入は現実的ではない．

したがって，LPWA 環境における効率的な運用を実現するためには，更新データ量を削減する更新方式に加え，メモリ制約の厳しい LPWA 通信モジュールでも動作する軽量な実行環境が必要である．

2. 関連研究

2.1 差分更新方式 bpatch

LPWA 環境における OTA 更新の効率化手法として，差分更新方式 bpatch が提案されている [1]．bpatch はファームウェア間の差分を COPY/ADD 命令として表現し，小規模な変更に対して高い圧縮率を実現する．一方で，差分生成には比較的高い計算資源を要すること，大規模な変更では差分サイズが増大すること，旧ファームウェアを保持するためのフラッシュ領域が必要となることなど，メモリ制約の厳しい LPWA 通信モジュールでは適用が難しい場合がある．

2.2 組込みスクリプト言語

組込み機器において運用中の動作変更や機能追加を容易にする手段として，組込みスクリプト言語の導入が広く検討されている．スクリプト言語を用いることで，ファームウェア全体ではなく動作ロジックのみを更新対象とでき，OTA 更新と比較して更新データ量を削減できる．

代表的な軽量スクリプト言語として，mruby/c や Lua が挙げられる．mruby/c は組込み用途を想定した実装であり，組込み環境への導入が容易である [2]．一方，Lua は小さな処理系として知られ，組込み用途で利用されることも多いが，実行時のメモリ使用量は用途や設定により増大し得ることが指摘されている [3]．

しかし，LPWA 通信モジュールのように搭載 RAM が極めて限られる環境では，これらの処理系であっても実行時メモリが制約となり，導入が困難となる場合がある．

3. 研究目的

本研究では，低速・狭帯域の LPWA 環境でも多種センサによる IoT 遠隔計測を実現するため，LPWA 通信モジュール上で動作する軽量スクリプト言語とその実行基盤（Virtual Machine, VM）を構築する．これにより，スクリプトを無線通信（LPWA）経由で送信するだけで，遠隔地から現地のモジュール動作を柔軟に変更できることを目指す．また，従来の OTA 更新と比較して更新データ量を削減し，限られた RAM 容量の範囲で実行可能であることを示す．

4. 研究内容

4.1 提案手法の概要

4.1.1 設計方針

本研究では，通信負荷の低減と動作変更の柔軟性を両立するため，更新対象をファームウェア全体ではなくスクリプトに限定する．具体的には，ベースファームウェアを固定し，LPWA 経由でスクリプトのみを更新することで，低速・狭帯域の LPWA 環境でも遠隔からモジュールの動作を変更可能とする．

また，多くの LPWA 通信モジュールは搭載メモリが極めて小さいため，既存の軽量スクリプト言語処理系よりもメモリ消費を抑えた VM 設計を目指す．具体的にはスタックマシン型を採用し，命令セットを LPWA 通信モジュールの計測用途に必要な機能に絞って構成する．

さらに，特定のハードウェアに依存しない汎用的な実行基盤の構築を目指す．将来的に CPU アーキテクチャや周辺回路が異なるモジュールを採用した場合でも，VM がハードウェアの差異を吸収することで，同一のスクリプトを再利用可能とする「ハードウェア抽象化」を重視した設計を行う．

4.1.2 システム構成

図 1 に示すように，本システムは変更不要な「ベースファームウェア」と LPWA 経由で更新可能な「スクリプト領域」から構成される．ベースファームウェアには通信制御やセンサ制御，データ送信に加えて，スクリプトを解釈・実行する VM を実装する．開発者はスクリプトを作成し，アセンブラでバイトコードに変換後，LPWA 経由でモジュールへ送信する．モジュール上の VM がバイトコードを解釈・実行することで，センサデータ取得や送信周期などの動作を遠隔から変更できる．

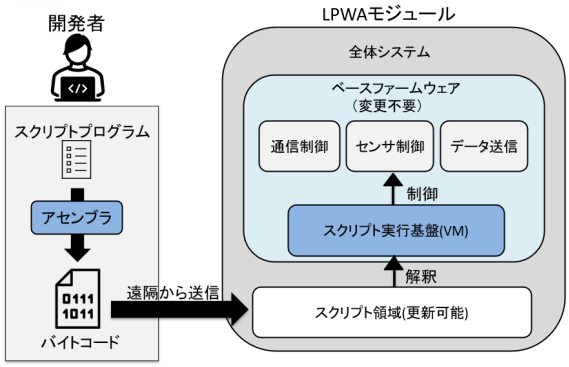

図 1　システム構成

4.1.3　スクリプト更新フロー

本研究におけるスクリプト更新の手順を以下に示す．

1. 開発者はスクリプトを作成する．
2. 更新対象スクリプトをバイトコードへ変換し，送信データ（Intel HEX 形式）として生成する．送信データには整合性確認用のチェックサムを含める．
3. 生成したバイトコードをモジュールに送信する（本実装では一行単位で送信する）．
4. モジュールは受信完了後，受信データに含まれるチェックサムと，モジュール側で計算したチェックサムを比較する．一致した場合のみ，受信したバイトコードをスクリプトとして保存する．不一致の場合は破棄し，保存を行わない．
5. スクリプト領域は最大 8 個のスクリプトを保持でき，更新時には指定された保存先に格納する．

4.2　提案言語と VM の仕様

4.2.1　VM 実行モデル

本研究で提案するスクリプト言語は，スタックマシン型の VM 上で実行される．VM はスクリプト（バイトコード）を 1 命令ずつ逐次解釈実行し，演算や I/O 操作をすべてスタック上の値の push/pop により実行する．本 VM は固定長のメモリ領域のみを使用し，動的メモリ確保を行わないことで，断片化を回避し予測可能な実行を実現する．

VM の情報

VM はプログラムカウンタ（PC），スタックポインタ（SP），ゼロフラグ（ZF）を保持する．PC は次に実行する命令位置を指し，SP はスタックトップを示す．ZF は算術演算結果が 0 であることを示す 1bit フラグであり，条件分岐命令で参照される．16bit 整数を基本単位とし，スタックおよび汎用メモリに格納する．スクリプトのロード時には PC と SP および ZF を 0 にリセットし，スクリプト先頭から実行を開始する．表 1 に VM の固定パラメータを示す．

命令実行（フェッチ・デコード・実行）

VM はまずバイトコード列から 1 バイトのオペコードを取得し PC を進める（フェッチ）．次に，オペコードに応じて即値や相対オフセット等のオペランドを読み出し（デコード），スタック操作，算術演算，メモリアクセス等の処理を実行する．算術演算命令（add/sub/mul/div/mod）では，スタックから 2 要素を pop し，結果を push する．演算結果が 0 の場合に ZF=1，それ以外は ZF=0 に更新する．

制御命令と相対ジャンプ

分岐命令（jmp/jz/jnz）は 8bit 相対オフセットをオペランドとして持ち，PC ← PC + offset により分岐する．jz は ZF=1 のとき，jnz は ZF=0 のときに分岐する．相対ジャンプを採用することで，バイトコード列をメモリ上の任意の位置に配置しても分岐先が変化せず，再配置可能（リロケータブル）なスクリプトを実現できる．

安全停止（エラーハンドリング）

実行中にスタックオーバーフロー／アンダーフロー，0 除算，無効メモリアドレスアクセスを検出した場合は，スクリプト実行を中断して VM を停止する．これにより，不正なスクリプトが実行された場合でも，通信制御を含む他の処理への悪影響を防ぐことができる．

表 1　VM の固定パラメータ

項目	値	備考
スタック要素型	int16_t	16bit 整数
スタックサイズ	256	要素数
汎用メモリ要素型	uint16_t	16bit ワード
汎用メモリサイズ	64	ワード数（=128 B）
ジャンプ方式	8bit 相対	符号付きオフセット

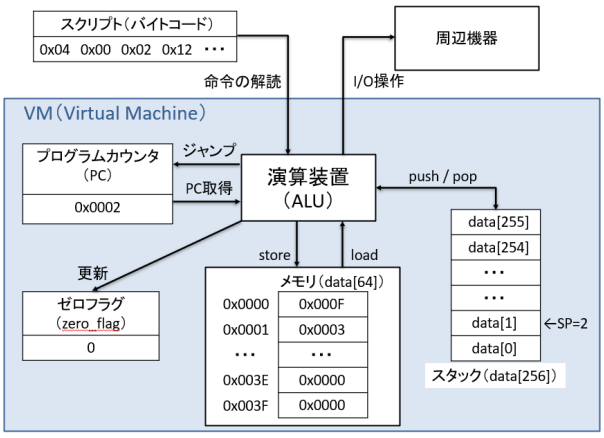

図 2　VM 実行モデル（状態とデータの流れ）

4.2.2　命令セット

本研究では，開発者の記述性と実行時のメモリ効率を両立するため，抽象化された「スクリプト命令（ニーモニック）」と，VM が直接実行する「バイトコード命令」を分離して設計した．アセンブラは，スクリプト命令を解析し，引数の値やジャンプ距離に応じて適切な実行形式（バイトコード）へと変換する．命令セット一覧を表 2 に示す．

表 2　命令セット一覧

スクリプト命令	VM 命令	opcode	説明
nop	NOP	0x00	何もしない.
halt	HALT	0x01	実行を終了する.
push <val>	PUSH_B	0x03	1 バイトの即値を積む.
	PUSH_W	0x04	2 バイトの即値を積む.
dup	DUP	0x05	スタックトップの値を複製する.
swap	SWAP	0x06	スタックトップ 2 要素を入れ替える.
drop	DROP	0x17	スタックトップを破棄する.
add	ADD	0x07	スタックから a, b を取り出し，$b + a$ を積む.
sub	SUB	0x08	スタックから a, b を取り出し，$b - a$ を積む.
mul	MUL	0x09	スタックから a, b を取り出し，$(b \times a)$ を積む.
div	DIV	0x0A	スタックから a, b を取り出し，(b/a) を積む.
mod	MOD	0x0B	スタックから a, b を取り出し，$(b \pmod a)$ を積む.
jmp <label>	JMP	0x0C	8bit 相対オフセットによる無条件分岐.
jz <label>	JZ	0x0D	ゼロフラグが 1 のときに相対分岐.
jnz <label>	JNZ	0x0E	ゼロフラグが 0 のときに相対分岐.
load	LOAD	0x12	スタックからアドレスを取り出し，値を積む.
store	STORE	0x13	スタックからアドレス，データの順に取り出し，書き込む.
wait	WAIT	0x15	指定時間待機する（ms 単位）.
bcd	BCD	0x16	値とアドレスを取り出し，ASCII 変換して格納.
print	PRINT	0x0F	スタックトップの値を出力する.

4.2.3　メモリマップド I/O

スクリプトから周辺機能を扱うため，メモリマップド I/O を採用する．周辺機能を特定アドレスへの load/store として抽象化し，スクリプト側は load/store 命令のみで I/O 操作を記述できる．これにより，ハードウェア依存の処理を VM 側に集約でき，将来モジュールが変更されてもアドレス対応の実装のみ差し替えることで同一スクリプトを再利用可能とする．表 3 に I/O アドレスを示す.

表 3　メモリマップド I/O アドレス

アドレス	アクセス	内容
0x0000–0x003F	load/store	汎用メモリ（64 ワード）
0x0100	store	LED 制御（0 で OFF，それ以外で ON）
0x0200	load	センサ ADC 値取得
0x1000	store	送信トリガ（バッファ先頭から送信）
0x1001–0x1010	store	送信用バッファ（各アドレスに 1 文字）

4.3　アセンブラ

4.3.1　アセンブラの役割

提案 VM はバイトコード列を逐次解釈して実行するため，テキスト形式のスクリプトを直接実行することはできない．そこで本研究では，Python を用いて独自のアセンブラを開発した．アセンブラは開発者側の PC で動作し，ソースコードをバイナリ形式のバイトコードへ変換する．これにより，リソース制約の厳しい端末側（LPWA 通信モジュール）には実行環境である VM のみを実装すればよく，デバイスの負荷を最小限に抑えられる．

4.3.2　変換規則と出力形式

本アセンブラは，スクリプト命令を対応するオペコードに変換し，必要に応じて即値や相対オフセットを付与してバイト列を生成する．本アセンブラには，LPWA 環境における送信データ量を削減するための最適化を導入している．具体的には，PUSH 命令において即値が 0〜255 の範囲内であれば 1 バイト即値（opcode: 0x03）として処理し，範囲外の場合のみ 2 バイト即値（opcode: 0x04）を選択する仕組みとした.

また，分岐命令（`JMP/JZ/JNZ`）には 8bit の相対オフセットを採用している．ラベル解決のために 2 パス方式を採用し，1 パス目で各命令の配置アドレスとラベル位置を記録し，2 パス目でラベル参照を具体的な相対オフセット（$-128 \sim 127$ の範囲）へと置換する.

最終的な出力は Intel HEX 形式とし，1 行単位で送信可能な形式に整形する．各行にはチェックサムを付与することで，LPWA 環境下での通信エラーによるデータの破損を受信側で検知可能にしている.

5.　評価実験

5.1　実験条件

評価実験には，EASEL 社の ES920LR3 ボード（LoRa 通信）を用いた [4]．通信パラメータは帯域幅 $BW = 500\,\mathrm{kHz}$，拡散率 $SF = 7$，符号化率 $CR = 1$ に固定した．文献 [5] に示されるビットレート表（理論値）に基づき，理論通信

速度は $R = 21.875\,\mathrm{kbit/s}$ とした．

5.2 通信負荷

5.2.1 実験内容

温度計測スクリプトについて，スクリプトサイズ（ヘッダを含むバイトコード）と理論通信速度 21.875 kbit/s に基づく送信時間（理論値，オーバーヘッド除く）を算出する．また，算出結果を他言語（方式）と比較する．送信時間 T は，通信が非同期シリアル（1 B あたりスタートビット 1 bit とストップビット 1 bit を含む）であることから 1 B = 10 bit とし，次式で算出した．

$$T\,[\mathrm{s}] = \frac{S\,[\mathrm{B}] \times 10\,[\mathrm{bit/B}]}{R\,[\mathrm{bit/s}]} \tag{1}$$

ここで，S はスクリプトサイズ（ヘッダを含むバイトコード），R は理論通信速度である．

5.2.2 実験結果

表 4　通信負荷の比較

対象	サイズ [B]	送信時間 (理論値) [ms]
提案言語	54	24.7
mruby/c	215	98.3
Lua	621	283.9

5.3 遠隔更新の所要時間

5.3.1 実験内容

スクリプトをバイトコードに変換して送信し，送信開始から更新完了 ACK を受信するまでの時間を 10 回測定して平均を求める．

5.3.2 実験結果

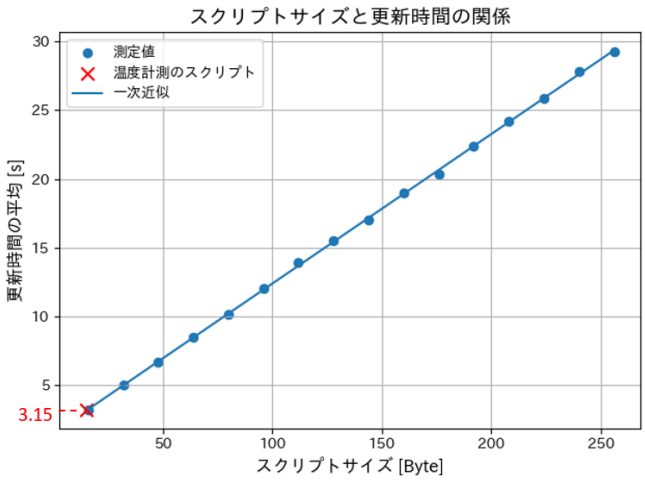

図 3　実験結果

図 3 の結果より，温度計測スクリプトでは平均約 3.15 秒で更新が完了し，更新時間はスクリプトサイズに概ね比例して増加することが分かった．

5.4 実行時 RAM 使用量

5.4.1 実験内容

スクリプト実行時に VM が確保する RAM 使用量を測定した．具体的には，VM 本体，スタック，汎用メモリ，送信用バッファの各領域について確保サイズを取得し，その合計を実行時 RAM 使用量とした．

5.4.2 実験結果

スクリプト実行時 RAM 使用量は，VM が静的に確保する固定領域の合計として 0.68 KB であった（VM:20 B，Stack:520 B，Memory:128 B，buf:16 B）．

5.5 到達性評価

5.5.1 実験内容

送受信機の距離を 500m〜5.0km の範囲で変化させ，各距離で 100 回受信し，受信成功率を評価する．

実験は見通しの良い環境と悪い環境の 2 条件で行った．図 4, 5 のように，見通しの良い環境として佐賀県白石町有明海沿岸，見通しの悪い環境として福岡県飯塚市で実験を行った．

図 4　見通しの良い環境（3.0km の例）

図 5　見通しの悪い環境（3.0km の例）

5.5.2 実験結果

表 5 到達性評価結果（試行回数 $n = 100$）

距離	500m	1.0km	2.0km	3.0km	4.0km	5.0km
見通しの良い環境	100%	100%	100%	100%	100%	100%
見通しの悪い環境	100%	98%	83%	83%	32%	35%

6. 考察

本研究では，更新対象をファームウェア全体ではなくスクリプトに限定することで更新データ量の削減を図り，LPWA 通信モジュール上で動作可能な軽量スクリプト言語とその実行基盤を設計・実装した．本章では，評価実験の結果に基づき，更新効率，RAM 使用量，および LPWA 通信の特性の観点から本方式の有効性と課題を考察する．

6.1 更新効率に関する考察

提案言語のスクリプトサイズ（54B）は，既存の軽量スクリプト言語である mruby/c（215B）や Lua（621B）と比較して，大幅に抑えられている．この軽量化は，低速・狭帯域という LPWA 通信の制約を克服する上で極めて有効であり，理論値に基づいた計算においても更新時間を大幅に短縮できると考えられる．

また，温度計測スクリプトの更新に要した時間は平均約 3.15 秒であり，これは運用中の動作変更において十分に実用的な更新時間と言える．

6.2 メモリ消費に関する考察

実験の結果，VM の実行に必要な RAM 使用量は 0.68KB に抑えられていることが確認された．多くの LPWA 通信モジュールは，RAM 容量が極めて限られている．この 0.68KB という極めて小さいメモリ消費はモジュールの主機能である通信制御やセンサ制御，データ送信の動作を妨げることなく，VM を共存させることが可能であることを示している．メモリ消費量を最小限に抑えられた主な要因は，スタックマシン型 VM の採用と，必要最低限の命令セットに特化した設計方針にある．命令解釈部を簡素化し，相対ジャンプやゼロフラグを用いた条件分岐を採用することで，制御用データのオーバーヘッドを削減した．この軽量性は，将来的にメモリ制約のある他のモジュールへ移植する際にも，大きな利点となると考えられる．

6.3 LPWA の通信特性と実運用における課題

到達性評価の結果，見通しの良い環境では 5.0km の長距離においても 100%の受信成功率を維持できた．一方で，見通しの悪い環境では距離に比例して成功率が低下し，5.0km 地点では 35%に留まった．この結果は，LPWA の長距離通信性能が障害物や地形の影響を強く受けることを示しており，実運用においては中継器の配置などにより見通しを確保する必要があると考えられる．

また，本システムにはチェックサムによる整合性確認が備わっているため，パケット欠落が生じた際に不完全なスクリプトが解釈・実行され，システムが不安定になる事態は回避される．しかし，通信環境が悪い地点では更新が完了しない可能性が高まり，更新の確実性に課題が生じる．そのため，更新完了の確認手段としてハッシュ値による検証を導入すること，および再送制御プロトコルを実装することが，システムの信頼性向上において重要であると考えられる．

7. 結言

本研究では，LPWA 環境に適した軽量スクリプト更新方式を提案し，スクリプト更新による遠隔機能変更と多種センサ計測の実現可能性を示した．評価実験より，提案方式はスクリプトサイズを 54 B まで削減でき，温度計測スクリプトの更新は平均約 3.15 s で完了した．また，スクリプト実行に必要な RAM 使用量は 0.68 KB であり，LPWA 通信モジュール上で動作可能であることを確認した．一方，到達性や更新完了の可否は通信環境の影響を受けるため，実運用では更新の確実性を考慮した設計が必要である．

今後は，ハッシュ値を用いたスクリプト更新の整合性確認，フリーズや無限ループ対策として Watchdog timer の導入，ならびにマクロ機能による開発容易性向上を検証する．

参考文献

[1] De Simone, A., Turvani, G. and Riente, F.: Incremental firmware update over-the-air for low-power IoT devices over LoRaWAN, *Internet of Things*, Vol. 34, p. 101772 (online), DOI: https://doi.org/10.1016/j.iot.2025.101772 (2025).
[2] ITOC しまねソフト研究開発センター：「mruby/c」，ITOC しまねソフト研究開発センター（オンライン），入手先 ⟨https://www.s-itoc.jp/support/technical-support/mrubyc/⟩（参照 2026-02-03）.
[3] Marinescu, B.: Reducing Lua memory footprint - need help, lua-l メーリングリスト（アーカイブ） (online), available from ⟨https://lua-l.lua.narkive.com/PLWqYHni/reducing-lua-memory-footprint-need-help⟩ (accessed 2026-02-10). 2008-10-12.
[4] EASEL: 920MHz 帯 LoRa/FSK モジュール（超低消費電力）ES920LR3, EASEL（オンライン），入手先 ⟨https://easel5.com/service/products-information/products/wireless-module/es920lr3/⟩（参照 2026-01-29）.
[5] 株式会社アールエフリンク：LoRa/GPS コンバータ・取り扱い説明書（RM-92XGPS141）Ver.1.0.4. PDF, 参照日: 2026-02-13, http://www.rflink.co.jp/pdf/RM-92X-GPS141/RM-92XGPS141-instruction$_manual$ − ver1.0.4.pdf.