小型IoT機器向けの移動エージェントフレームワークの提案と実装

坂本 和也[†1]　中屋 悠資[†2]　北形 元[†3]　長谷川 剛[†3]

概要：LPWA を用いた IoT ネットワークは災害対策のための環境情報の取得など，さまざまな場所で活用されている．通信容量の小さい LPWA を用いるうえで，通信量の抑制は重要である．加えて，災害対策のための環境情報の取得などにおいては，デバイスが外部の状況に応じて適応的に動作することも求められる．WSN (Wireless Sensor Network) の分野では，通信量を抑制するために，移動エージェントを用いた手法が提案されている．移動エージェントを用いてデータ処理を行うことで，通信量の削減に加えて，移動しながら動作することにより IoT 機器の動作をオンデマンドで柔軟に変更できると考えられる．しかし，これらの研究においては，具体的な移動エージェントの実装は示されていない．そこで本稿では，IoT 機器上でデータ処理を実行し，状況に応じて IoT 機器の動作を変化させるための，移動エージェントフレームワークを提案する．必要最小限の機能で構成することにより，実装におけるメモリ消費量を小さくする．さらに，エージェントの移動とメッセージ送信に対し，それぞれの特徴を考慮した圧縮法を導入し，エージェント間の通信量を削減する．実装実験の結果，提案するフレームワークは約 450 KB のメモリ消費量で動作することが分かった．また，エージェントの移動時の通信量を平均で 53.9%小さくできることが分かった．さらに，センサデータの送信を想定した実験により，既存手法と比較して通信量を平均で 93.5%削減できることが分かった．

1. はじめに

IoT 機器を用いた屋外の環境情報のモニタリングなどにおいて，IoT 機器を電源供給のない状態で長時間稼働させる場合には，機器自体が低消費電力で動作することに加えて，通信により消費される電力が小さいことが望ましい．さらに，少ない IoT 機器で広範囲をモニタリングするには，モニタリング能力が広範囲である必要があり，さらに，通信によってその情報を収集する場合には通信距離が長いほうが良い．そこで，低消費電力と長距離通信を兼ね備えた通信方式として LPWA (Low Power Wide Area) が注目されている [1]．LPWA を用いた IoT 機器で構成されるネットワーク (以降，LPWA を用いた IoT ネットワークと表記) は，農業における環境情報の取得や [2] や LP ガスのスマートメーターによる検針 [3] など，さまざまな場所で活用されている．また，LPWA を用いたマルチホップ通信も提案されており [4]，シングルホップ通信と比較して，より広域なネットワークを構成しやすくなるため，IoT 機器を用いた広域センシングへの応用が期待できる．

しかし，LPWA の通信速度は，WiFi や Bluetooth 等の自営網を構築可能な他の無線通信方式に比べて遅い．さらに，マルチホップ通信でデータを収集する際には，データを集積したり，インターネットとのゲートウェイとなる中央ノード付近にトラフィックが集中するため，通信速度の遅さが顕著な問題となる．また，例えば災害対策のための環境情報のセンシングにおいては，地震の震度や雨量などに異常値が検出された際にそのデータを優先的に送信することや，周囲の環境変化に応じてデータの集約や統計処理方法を切り替えるなど，適応的なデータ処理が求められる．しかし，あらゆる未知の状況に対応するプログラムを予め機器に搭載しておくことは困難である．

WSN (Wireless Sensor Network) の分野では，ネットワークの経路上でデータを処理して通信量を削減する研究がなされており [5][6][7]，その中に，移動エージェントを用いたデータ融合により通信量を削減する研究がある [7]．移動エージェントを用いることで，データ処理を実行して通信量を削減することに加えて，ノード間を移動して自律

†1 現在，東北大学　大学院情報科学研究科
Presently with Graduate School of Information Sciences, Tohoku University
†2 現在，東北大学　大学院工学研究科
Presently with Graduate School of Engineering, Tohoku University
†3 現在，東北大学　電気通信研究所
Presently with Research Institute of Electrical Communication, Tohoku University

的に動作することにより，IoT 機器の挙動をオンデマンドに変更することができる．しかし，[7] では，シミュレーションによる評価は行われているが，IoT 機器で稼働する具体的な移動エージェントの実装は示されていない．

そこで本稿では，LPWA を用いた IoT ネットワークにおいて，IoT 機器上で周囲の環境変化に応じた適応的なデータ処理を実現するための，小型 IoT 機器向けの移動エージェントフレームワークを提案する．既存の移動エージェントフレームワークでは，メモリ消費量などの制約により，小型 IoT 機器上で動作させることができないため，必要最小限の機能のみを用いて構成することで，メモリ消費量を小さくする．さらに，エージェントの移動とメッセージ送信に対し，それぞれの特徴を考慮した圧縮法を導入し，エージェント間の通信量を削減する．また，提案するフレームワークに基づく移動エージェントの実行環境の設計と実装を行い，IoT 機器を用いて実験と評価を行う．

2. 関連研究

ここでは WSN の分野で移動エージェントを用いた研究について述べる．Jin-Gang Cao の研究 [7] では，WSN のエネルギー消費と遅延に着目し，移動エージェントの技術を適用したデータ融合のためのルーティングアルゴリズムを提案している．この提案では，改良された蟻コロニー最適化アルゴリズムを用いて移動エージェントの経路探索をしている．シミュレーションを用いて，提案されたアルゴリズムが遅延を減らしネットワーク寿命を延ばすことが示されている．しかしながら，具体的なエージェントの実装を用いた評価については述べられていない．また，梅澤らの研究 [8] では，移動エージェントを用いたセンサネットワーク向けフレームワークを提案している．彼らは，センサネットワークを用いたビル内のユーザの位置検出において，所定のフロアにいるユーザを常時監視する場合と，特定のユーザの位置を検出する場合で要求される機能が異なると指摘している．そこでセンサノードに提案したフレームワークを実装し，移動エージェントを用いて，センサネットワークを利用するアプリケーションの要求変化に応じて，センサノードの機能を変更している．しかしながら，実装には Windows2000 および JDK1.1 が稼働する PC を用いており，IoT 機器のようなメモリの小さい機器への実装については述べられていない．

3. 小型 IoT 機器向けの移動エージェントフレームワーク

前章で述べたように，WSN の分野で移動エージェントを用いた研究はされているが，IoT 機器への実装は示されていない．そこで，小型の IoT 機器上で移動エージェントを動作させるための移動エージェントフレームワーク（以降，提案フレームワークと表記）を提案する．

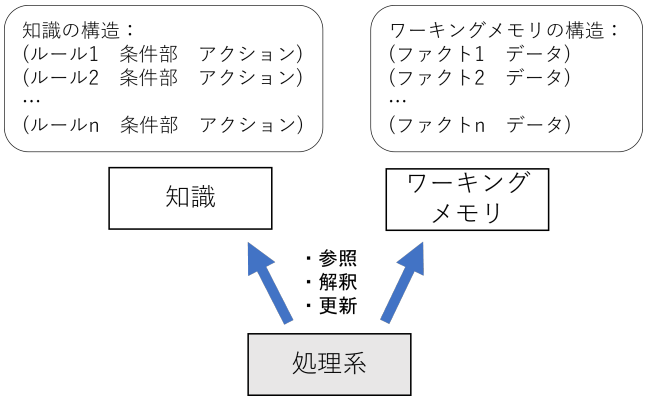

図 1: ルール型の知識処理に基づくエージェントの動作決定の仕組み

3.1 節，3.2 節で本研究で用いる移動エージェントとエージェント間通信言語について述べた後，3.3 節以降で提案フレームワークの概要について述べる．

3.1 移動エージェント

移動エージェントとは，ネットワークに接続された機器間を移動しながら，計算処理を継続していく自律的なソフトウェアである [9]．本研究では既存のエージェントフレームワークである DASH[10] で用いられているエージェントを参考とする．移動エージェントはセンサや他のエージェントからのメッセージを通して外部からの情報を受け取る．そして，その情報をもとに次にとる動作を決定する．その後，その決定に従い実際の動作を行う．図 1 にルール型の移動エージェントの動作決定の仕組みを示す．知識はエージェントの動作記述であり，条件部と実行部 (アクション) からなる IF-THEN 型のルールの集合である．ワーキングメモリはファクトと呼ばれるエージェントの保持する情報やセンサ等から得た情報の集合である．処理系はそれらを参照し，条件部がワーキングメモリのデータとマッチするルールを探し，そのルールのアクションを実行，すなわち動作を決定し実行する．

3.2 エージェント間通信言語

エージェント間通信言語 (ACL: Agent Communication Language) とは，エージェント間でやり取りされる情報の構文や，意味内容を規定しているもので，エージェントはこの ACL に基づくメッセージ通信によって情報交換や協調動作を行う．代表的な ACL として FIPA ACL[11] がある．これは，エージェント技術標準化団体である FIPA(Foundation for Intelligent Physical Agents) によって発表された ACL であり，JADE[12] 等のエージェントプラットフォームで採用されており，本研究で用いるエージェントでは FIPA ACL を参考にする．

ACL では，送信するコンテンツ以外に，送信元や宛先

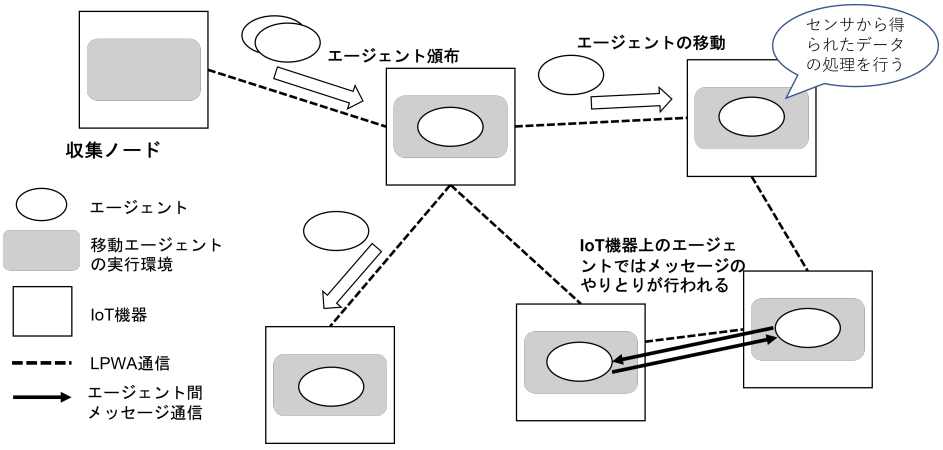

図 2: 提案フレームワークを用いた IoT ネットワーク

のエージェント名等の付加情報を含む．例として，温度センサから得たデータを送信する場合のメッセージを下記に示す．

```
:performative :inform-temp :to agent-b :from
agent-a :content ( temp :unit degree :value
17.2)
```

先頭にコロンが付与された単語は属性を示しており，属性の次に来る単語，あるいはリストがその属性に対応する値である．`:performative` はメッセージの種類 (パフォーマティブ)，`:to` は宛先のエージェント名，`:from` は送信元のエージェント名，`:content` は伝える内容 (コンテンツ) を表す属性で，全てのメッセージで共通する予約語である．一方，`:unit` や`:value` はエージェントの開発者が任意に定める属性である．

このようにエージェントのメッセージには，伝える内容だけでなく付加情報が必要であるため，送信したい温度の情報 (コンテンツ) 数バイトに対して，メッセージ全体のサイズは数十バイトになる．

3.3 提案フレームワークの概要

本稿での提案フレームワークは，小型の IoT 機器上で移動エージェントを動作させるための実行環境とその上で稼働する移動エージェントを表す．提案フレームワークは，既存のフレームワークから必要最小限の機能を抽出して構成することで，実行環境のメモリ消費量を小さくし，小型の IoT 機器での動作を可能とする．また，LPWA の利用を想定し，エージェントの移動の際の通信量やエージェント間メッセージの送受信サイズを削減する．

図 2 に提案フレームワークを用いた IoT ネットワークの全体図を示す．各 IoT 機器は LPWA を用いてネットワークを構成しており，移動エージェントの実行環境は各 IoT 機器に内包されている．エージェントは実行環境の間を自律的に移動し，動作する．これにより，必要な処理プログラムを搭載したエージェントが目的の機器まで移動し，周

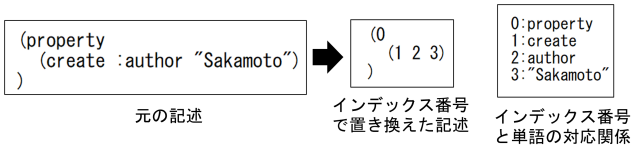

図 3: 移動エージェントの省サイズ化を行った例

囲の状況変化に応じて柔軟にデータを処理することができる．

3.4 必要最小限のエージェントアクションの導入

エージェントのアクションを多数導入すると多様な動作が可能になるが，実行環境のプログラムサイズとエージェントを実行したときのメモリ消費量は増加する．そのため，提案フレームワークでは，既存のフレームワークから必要最小限のアクションを抽出して導入することで，軽量化する．提案フレームワークにおいて，移動エージェントは 3.1 節で述べたように，ワーキングメモリと知識を参照して動作を決定する．ワーキングメモリの内容はセンサから得た情報や他のエージェントから受信したメッセージ等，外部からの情報が反映される．そのため，ワーキングメモリの操作を行うアクションが必要となる．そこで，ワーキングメモリに対してファクトの追加，削除，更新を行うアクションとして，それぞれ `make`，`remove`，`modify` を導入する．また，エージェントは各機器上の実行環境の間を移動をしたり，エージェント間で ACL を用いたメッセージ通信を行う．そのため，通信を行うアクションが必要となる．そこで，`send`，`move` の 2 つのアクションを導入する．`send` はメッセージの送信，`move` は動作中のエージェントの移動を行うアクションである．さらに，センサから取得したデータ等を加工するために，手続き型の処理を実行するアクションが必要となる．そこで，手続き型言語の形式で記述されたプログラムを実行するアクションとして，`control` を導入する．

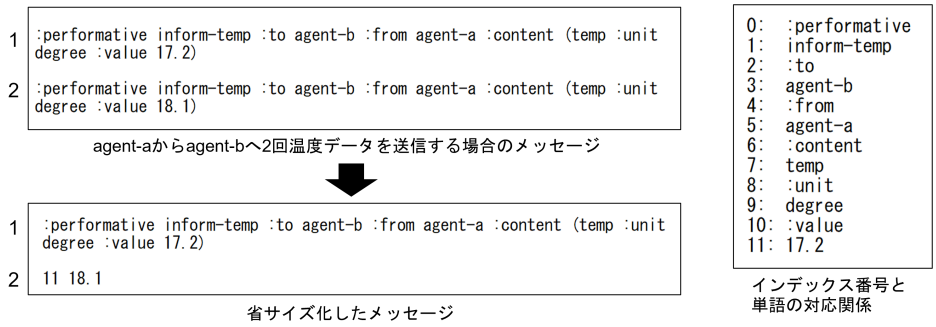

図 4: エージェント間通信メッセージの省サイズ化を行った例

3.5 移動エージェントの省サイズ化

エージェントの移動は，エージェントの動作記述（以降，エージェント記述と表記）とワーキングメモリの内容を移動先の実行環境へ送信することで実現する．一般に，エージェント記述では，同じ単語が複数回使用されることが多い．そこで，エージェント記述に使用されている単語をインデックス番号に置き換えることにより，エージェント記述のサイズを小さくし，移動の際の通信量を削減する．以下にその手順を示す．

1. エージェント記述に使用されている単語を抽出してインデックス番号を付与し，単語とインデックス番号からなる辞書を作成する．
2. エージェント記述内の単語をインデックス番号に置き換える．
3. 作成した辞書を用いてワーキングメモリ内の単語をインデックス番号に置き換える．
4. 1. から 3. で作成した辞書とエージェント記述，およびワーキングメモリの内容を送信する．

図 3 に上記の手順を用いて，移動エージェントの省サイズ化を行った例を示す．

また，単語の辞書について，すべてのエージェントに共通する予約語など，使用頻度の高いものを共通辞書としてあらかじめ各実行環境に内蔵させておくことで，1. で作成する辞書のサイズを小さくし，4. で辞書を送信する際の通信量を削減する．受信側では，あらかじめ内蔵させた予約語の辞書と新たに受信した辞書を参照して，インデックス番号で置き換えられたエージェント記述とワーキングメモリを復元する．

3.6 エージェント間通信メッセージの省サイズ化

3.2 節で述べたように，ACL では付加情報によりメッセージサイズが大きくなる．特に，同じエージェント間でセンサデータを定期的に送信するような場合，メッセージの大部分は毎回同じである．そこで，最後に送信したメッセージと比較し，差分のある部分のみを送信することで，送信メッセージのサイズを小さくし通信量を削減する．

まず，送信側では最後に送信したメッセージを送信済みメッセージとして記録しておく．受信側でも同様に，最後に受信したメッセージを記録しておく．そして，下記の手順で送信するメッセージを省サイズ化する．

1. 送信するメッセージを単語ごとに分解し，先頭から順にインデックス番号を割り振る．
2. 送信するメッセージと記録してある送信済みメッセージの単語を順に比較する．
3. 比較した結果，両者で単語が異なる場合，送信するメッセージからその単語を抽出する．
4. 抽出した単語とそれに対応するインデックス番号のみを送信する．

図 4 に，上記の手順を用いて，agent-a から agent-b へ 2 回温度データを送信する際のメッセージの省サイズ化を行った例を示す．1 回目のメッセージと 2 回目のメッセージの違いは，インデックス番号 11 の，温度を表す値のみである．よって，2 回目のメッセージを送信する際は，インデックス番号である 11 と，変化した値である 18.1 のみを送信する．受信側では，記録してあるメッセージを参照して，受信したインデックス番号に対応する単語を受信した単語に置き換えることで，メッセージを復元する．また，メモリの使用量を抑えるため，エージェントが移動する際には記録されているメッセージを削除する．

4. 設計

4.1 全体構成

図 5 に提案フレームワークの全体構成を示す．フレームワークは，エージェントと実行環境から成る．さらに，実行環境はエージェント処理機能，通信機能，I/O 機能から構成される．

- エージェント処理機能
  エージェント記述に基づきエージェントを動作させる機能であり，後述するエージェントインタプリタが該当する．

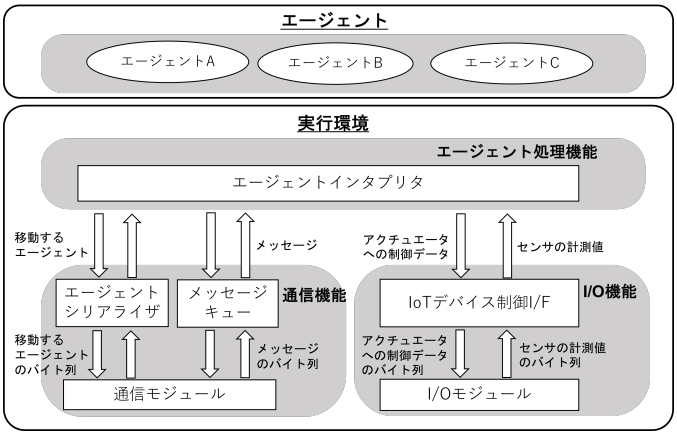

図 5: 提案フレームワークの全体構成

- 通信機能
  エージェント間の通信やエージェントの移動を実現する機能である．エージェントシリアライザ，メッセージキュー，および通信モジュールが該当する．エージェントシリアライザは，動作中のエージェントを他の実行環境へ移動させるために，バイト列に変換する．この際に前章で述べた通信メッセージの省サイズ化を行う．メッセージキューは，送受信されるメッセージを一時的に保管する役割を持ち，さらに前章で述べたエージェント間メッセージの省サイズ化を行う．通信モジュールは，LPWA 等を用いて他の IoT 機器と通信する．
- I/O 機能
  センサからデータを入力したり，アクチュエータへデータを出力する機能である．IoT デバイス制御 I/F(インターフェース) と，I/O モジュールが該当する．IoT デバイス制御 I/F は，近年の IoT 機器が具備しているニューラルネットワーク用の演算アクセラレータ [13] や，論理回路をソフトウェアで構成/再構成可能な FPGA[14] 等の特別な機能を使用する際のインターフェースとしての役割を持つ．I/O モジュールは IoT 機器が具備する GPIO (General-Purpose Input/Output)，ADC (Analog-Digital Converter) /DAC (Digital-Analog Converter) 等の入出力機能，および，UART (Universal Asynchronous Receiver/Transmitter)，I2C (Inter-Integrated Circuit)，SPI (Serial Peripheral Interface) 等のシリアル通信機能を使用し，センサからのデータの読み取りやアクチュエータの操作を行う．

4.2 エージェントインタプリタ

エージェントインタプリタは，稼働するエージェントの記述を解析しながら，知識とワーキングメモリを更新する．具体的には，知識内の各ルールの条件部にマッチするファクトをワーキングメモリ内から検索し，ルールに記述されたアクションを実行することでエージェントを動作させる．また，4.1 節で述べたように，他のエージェントとの通信の際には通信機能を利用し，センサやアクチュエータを使用する際には I/O 機能を利用する．

4.3 ファクトの表現形式

3.1 節で述べたように，ワーキングメモリはファクト呼ばれるエージェントの保持する情報の集合である．ファクトは OAV 形式 [10] で表現する．OAV 形式のファクトの記述例を下記に示す．

```
(object :attribute1 value1 :attribute2 value2)
```

OAV 形式では，ファクトの種類を表すオブジェクト名 object と，複数の属性 attribute とその値 value のペアを格納できる．

4.4 エージェントの記述形式

図 6 にメッセージを送信するエージェント記述の例を示す．エージェント記述は以下の要素からなるリスト構造から成る．

- エージェント名
  エージェントの名前．通信時の宛先等に使用する．
- (property <属性 1> ...<属性 n>)
  エージェント固有の情報を記述した属性のリスト．各属性は OAV 形式で記述する．ワーキングメモリにファクトとして追加され，更新はできない．
- (initial_facts <初期状態 1> ...<初期状態 n>)
  エージェントを初期化する際の情報を記述した初期状態のリスト．各初期状態は OAV 形式で記述する．属性記述リストと同様に，ワーキングメモリにファクトとして追加されるが，更新が可能である．
- (knowledge <ルール記述 1> ...<ルール記述 n>)
  エージェントの知識であるルール記述のリスト．1 つ以上のルール記述から構成される．ルール記述はルール名，条件部，実行部から構成され，以下のように表現される．
  (rule <ルール名> <条件部> --> <実行部>)

5. 実装と評価

5.1 実装

4 章で述べた設計に基づき，提案フレームワークの試作システムを実装した．試作システムの実装には，IoT 機器用のマイコンとして Espressif Systems の ESP-WROOM-32[15], 通信モジュールとして 920MHz 帯を使用する LPWA モジュールである Interplan の IM920s[16] を用いた．実行環境の実装には，インタプリタである MicroPython[17] を用いた．MicroPython は Python3 の標準ライブラリのサブセットを含む実装であり，マイコンなどの制約のある環境で実行できるように小型化・最適化されている．インタ

```
(agent Sample
  (property
    (create :author "Sakamoto")
  )
  (initial_facts
    (worker :name "Tanaka")
  )
  (knowledge
    (rule author
      (Msg :performative question :from ?
        from :content ("author?")) = ?msg
      (create :author ?a)
      -->
      (send :performative information :to
        ?from :content (author ?a))
      (remove ?msg)
    )
    (rule sorry
      (Msg :performative != __INIT_C) = ?
        msg
      -->
      (reply :performative sorry :to ?msg
        :content ?msg)
      (remove ?msg)
    )
  )
)
```

図 6: メッセージを送信するエージェント記述の例

プリタを用いることで IoT 機器のハードウェア構成への依存性が低くなる．

提案フレームワークの実行環境自体の記述量をできるだけ小さくするため，エージェント記述の構文解析には，MicroPython が持つ正規表現の機能を活用した．ただし，複雑な正規表現を使用すると，実行時のメモリ消費量が増加するため，一度の正規表現の適用でエージェント記述内の必要な情報を全て抽出するのではなく，単純な正規表現を段階的に適用して情報を抽出することで，メモリ消費量を抑制した．例えば，図 6 の property 内のファクトを抽出する場合，property から始まるリストをまず抽出し，その後，取り出したリストに含まれるファクトを抽出するようにした．

通信機能に関しては，IM920s をシリアル通信により制御し，LPWA による通信を行うようにした．

I/O 機能に関しては，ESP-WROOM-32 の GPIO を使用し，接続されたセンサからデータを受け取れるようにした．

5.2 実験環境

本実験では，5.1 節で実装した試作システムを用いる．また，IoT 機器にプログラムを転送したり，IoT 機器から文字出力をモニタリングするために，PC (Windows10) を 1 台用いる．PC と各 IoT 機器は USB ケーブルで接続した．また，IoT 機器とシリアル通信を行うために，PC 用のターミナルソフトとして TeraTerm[18] を使用した．

表 1: IoT 機器のメモリ消費量

	メモリ消費量 (KB)
MicroPython	409.824
MicroPython+実行環境	427.552
MicroPython+実行環境+移動エージェント	438.048

5.3 実験 1：メモリ消費量の評価

実装した試作システムのメモリ消費量を計測し，どの程度のメモリを搭載した IoT 機器であれば，提案フレームワークの実行環境が動作可能であるか確認する．

5.3.1 実験手順

1. IoT 機器で MicroPython を実行する．
2. メモリ消費量を計測するため，micropython クラスの mem_info() メソッドを使用して，メモリ消費量を計測する．
3. 実行環境を動作させ，同様にメモリ消費量を計測する．
4. 実行環境に移動エージェントを転送して実行し，同様にメモリ消費量を計測する．

5.3.2 実験結果

表 1 に計測したメモリ消費量を示す．表 1 の結果より，IoT 機器上で MicroPython を実行したときに 409.824 KB，その上で実行環境を動作させたときは 428.912 KB，さらに移動エージェントを転送し，実行したときのメモリ消費量は 438.048 KB であった．本実験では，温度を計測して送信するという機能を備えたエージェントを使用した．

以上の結果より，提案フレームワークの実行環境は，おおよそ 450 KB 以上のメモリを搭載した IoT 機器で動作可能であることが分かる．

5.4 実験 2：移動エージェントの省サイズ化の効果の確認

移動エージェントの省サイズ化の効果を確認するため，汎用の圧縮方法である zlib 圧縮を用いる場合と提案する圧縮方法を用いる場合の，圧縮率を比較する．本実験では，圧縮対象として我々の研究グループで過去に作成したエージェントを使用した．サンプル数は 2724 である．

5.4.1 実験手順

1. zlib 圧縮機能を追加した MicroPython を IoT 機器上で実行し，メモリ消費量を計測する．
2. 移動エージェントの元のサイズを計測する．
3. 移動エージェントに省サイズ化を適用し，省サイズ化

表 2: zlib 圧縮機能の追加に伴う IoT 機器のメモリ消費量

	メモリ消費量 (KB)
オリジナルの MicroPython	409.824
zlib 圧縮機能を追加した MicroPython	456.000

後のサイズを計測する．

4. 移動エージェントに zlib 圧縮を適用し，圧縮後のサイズを計測する．

5. 移動エージェントに省サイズ化と zlib 圧縮の両方を適用し，圧縮後のサイズを計測する．

6. すべてのエージェントに対して 2.〜5. を行う．

5.4.2 実験結果

表 2 に zlib 圧縮機能を追加した MicroPython を実行した際の，MicroPython のメモリ消費量を示す．zlib 圧縮機能を追加する前と比較して，約 50 KB メモリ消費量が増加した．

図 7 に，移動エージェントの圧縮前のサイズに対する圧縮率を示す．図 7 の横軸は移動エージェントの圧縮前のサイズ，縦軸は移動エージェントの圧縮前のサイズを区間 $1.5^n (n = 1, 2, \ldots)$ バイトで区切った各区間の平均の圧縮率を示している．圧縮率は (圧縮後のサイズ)/(圧縮前のサイズ)，すなわち，圧縮前のサイズを 1 としたときの圧縮後のサイズを表している．図 7 の結果より，省サイズ化を適用すると平均で 53.9%小さくできることが分かった．また，圧縮前のサイズが 10^3 バイト付近を境にして，提案手法と zlib 圧縮の圧縮率の大小関係が逆転することが分かった．すなわち，圧縮前のサイズがおおよそ 10^3 バイトから小さくなるほど提案手法の圧縮率が良く，圧縮前のサイズが大きくなるほど zlib 圧縮の圧縮率が良い．これは，圧縮前のサイズが小さいほど全体に対する予約語の割合が増えるため，予め共通辞書を持っている提案手法の方が有利になるためと考えられる．さらに，提案手法と zlib 圧縮を組み合わせたものでは，ほぼすべての区間において zlib 圧縮単体より圧縮率が優れていることが分かった．

5.5 実験 3：エージェント間通信メッセージの省サイズ化の効果の確認

エージェント間通信メッセージの省サイズ化の効果を確認するため，気温や湿度などのセンサデータを送信する際の通信量について，既存の ACL で記述されたメッセージをそのまま送信する場合と提案手法を適用した場合の，送信メッセージサイズを比較する．

5.5.1 実験手順

1. パフォーマティブが 1 種類のメッセージを，既存の ACL 形式で，送信間隔を 1 分として 24 時間送信し，

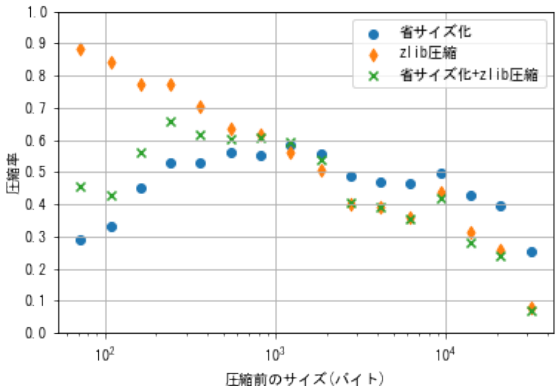

図 7: 移動エージェントの圧縮前のサイズに対する圧縮率

表 3: メッセージを送信間隔 1 分で 24 時間送信するのに要する通信量

	パフォーマティブの種類	通信量 (バイト)
元のメッセージ	1	132,139
	2	131,599
	3	129,031
省サイズ化後のメッセージ	1	8,384
	2	8,653
	3	8,330

通信量を計測する．

2. 1. と同様のメッセージを省サイズ化して，送信間隔を 1 分として 24 時間送信し，通信量を計測する．

3. メッセージのパフォーマティブを 2 種類，3 種類として 1.〜2. を行う．

5.5.2 実験結果

表 3 にメッセージを送信間隔 1 分で 24 時間送信するのに要する通信量を示す．表 3 の結果より，通信量は平均で 93.5%削減できたことが分かる．以上の結果より，センサデータを繰り返し送信する場合，提案したメッセージの省サイズ化により，通信量を削減できることを確認した．

5.6 評価

実験 1 では，提案フレームワークの試作システムが約 450 KB のメモリ消費量を搭載した IoT 機器で動作可能であることを示した．これにより，提案フレームワークが具体的な IoT 機器で動作可能であることを示した．WSN のセンシングに関する研究では，500KB 程度のマイコンが使用されている例があり，提案フレームワークはそれらのマイコンで動作可能であると考えられる [19][20]．実験 2 では，提案手法によりエージェントの移動に関わる通信量を削減できることを示した．また，zlib 圧縮との比較では，提案手法は実装のメモリ消費量は小さくなるが，エージェントのサイズが大きくなると zlib 圧縮より圧縮率が低くなるこ

とが分かった．これにより，想定する IoT 機器のメモリ量やエージェントのサイズによって，提案手法と zlib 圧縮機能でどちらを使用するか判断することができると考えられる．実験 3 では，提案手法によりエージェント間メッセージの送受信に関わる通信量を削減できることを示した．

6. まとめと今後の課題

6.1 まとめ

本稿では，LPWA を用いた IoT ネットワークにおいて，IoT 機器上で周囲の環境変化に応じて適応的なデータ処理を実現するための，小型 IoT 機器向けの移動エージェントフレームワークを提案した．

WSN の分野において，移動エージェントを用いたデータ融合により通信量を削減する研究が存在するが，メモリの小さい小型 IoT 機器への移動エージェントの実装については述べられていない．そこで本稿で提案するフレームワークでは，必要最小限の機能のみを用いて構成することで，実行環境のメモリ消費量を小さくし，小型 IoT 機器での移動エージェントの実行を可能にした．また，LPWA の利用を想定し，エージェントの移動の際の通信量やエージェント間メッセージの送受信サイズを削減した．試作システムを用いた実験の結果，提案フレームワークの実行環境は約 450 KB のメモリを搭載した IoT 機器で動作可能であることを確認した．

また，エージェントの移動時の通信量を平均で 53.9%小さくできることを示した．さらに，センサデータの送信を想定した実験により，既存手法と比較して通信量を平均で 93.5%削減できることを示した．

6.2 今後の課題

今後は，性能の異なるマイコンを対象として評価を行い，提案手法の適用範囲をより明確にしていく予定である．また，LPWA によるマルチホップ通信と組み合わせて，複数台の IoT 機器を用いた具体的なアプリケーションを想定した環境で，実験，評価をしていく予定である．

参考文献

[1] Zhihao Zhang, Baiguo Zhang and Xiaobo Zhang. : Performance Research of LoRa at High Transmission Rate, Journal of Physics:Conference Series, Vol.1544, p.012177 (2020).

[2] 日本データ通信：LPWA が拓く地域の IoT 社会：KDDI における LPWA の活用事例，(入手先 ⟨https://www.dekyo.or.jp/info/2019/01/seminar/5499/⟩) (2021-06-20).

[3] NEC：LP ガスメーター指針値提供サービス，(入手先 ⟨https://jpn.nec.com/lpgasmdms/index.html⟩) (2021-06-20).

[4] 佐藤剛至，柴田義孝，内田法彦：LoRa フラッディング技術を活用した路車間通信システムの開発と検証，信学技報，Vol.118，No.465，pp.511–514 (2019).

[5] Chi-Tsun Cheng, Henry Leung and Patrick Maupin. : A Delay-Aware Network Structure for Wireless Sensor Networks With In-Network Data Fusion, IEEE Sensors Journal, Vol.13, No.5, pp.1622-1631 (2013).

[6] Zhang Lin, Huan-Chao Keh, Ruikun Wu and Diptendu Sinha Roy. : Joint Data Collection and Fusion Using Mobile Sink in Heterogeneous Wireless Sensor Networks, IEEE Sensors Journal, Vol.21, No.2, pp.2364-2378 (2021).

[7] Jin-Gang Cao. : A data fusion routing algorithm in wireless sensor network based on mobile agent, Proc. In 2013 International Conference on Machine Learning and Cybernetics, Vol.01, pp.1-4 (2013).

[8] 梅澤猛，佐藤一郎，安西祐一郎：モバイルエージェントを用いたセンサネットワーク向けフレームワーク，情報処理学会論文誌，Vol.44, No.3, pp.779-788 (2003).

[9] 本位田真一：モバイルエージェントのためのソフトウェアプラットフォーム，NII Journal，No.3，pp.67–68 (2001).

[10] 原英樹：Dash マニュアル，(入手先 ⟨http://uchiya.web.nitech.ac.jp/idea/html/index.html⟩) (2021-06-28).

[11] Foundation for Intelligent Physical Agents：Welcome to FIPA!，(入手先 ⟨http://www.fipa.org⟩)(2021-07-04).

[12] Bellifemine, Fabio, et al. : JADE - A Java Agent Development Framework, Multi-Agent Programming. Springer, pp.125-147 (2005).

[13] Sipeed，(入手先 ⟨https://www.sipeed.com/⟩)(2021-06-28).

[14] Arduino mkr vidor 4000，(入手先 ⟨https://store.arduino.cc/usa/mkr-vidor-4000⟩) (2021-06-28).

[15] Espressif Systems：ESP32 Series of Modules，(入手先 ⟨https://www.espressif.com/en/products/modules/esp32⟩) (2021-07-07).

[16] Interplan：マルチホップ対応 920MHz 無線モジュール IM920s，(入手先 ⟨https://www.interplan.co.jp/solution/wireless/im920s/im920s.php⟩) (2021-06-28).

[17] MicroPython - Python for microcontrollers，(入手先 ⟨https://micropython.org/⟩) (2021-06-28).

[18] Tera term open source project，(入手先 ⟨https://ttssh2.osdn.jp/index.html.ja⟩) (2021-06-28).

[19] Junhui Cheng, Jianxun Jin, Bin Wang and Yourui Huang. : Motor Mperating State Monitoring System Design Using Wireless Sensor Network, 2020 IEEE International Conference on Applied Superconductivity and Electromagnetic Devices (ASEMD), pp.1-2(2020).

[20] Yang Yang. : Design and Application of Intelligent Agriculture Service System With LoRa-based on Wireless Sensor Network, 2020 International Conference on Computer Engineering and Application (ICCEA), pp.712-716 (2020).