Vol.2021-DPS-187 No.31
Vol.2021-MBL-99 No.31
Vol.2021-ITS-85 No.31
2021/5/28

木構造を拡張したセンサー識別子の自動割当

高木 優希[1,a)]　串田 高幸[1,b)]

概要：IoT デバイスに取付けられたセンサーは，識別可能な情報を保持しない場合が多く，実測値のみを出力している．本研究では木構造におけるノード同士の通信を可能とした新たなセンサー識別子を生成し，センサーに割当を行なった．識別子は不変的かつ衝突率を最小にするため，登録日時からハッシュによる生成を行なった．また親ノードの変更に伴う識別子の再生成が木構造により不要となる．実験はシステムの拡張性を評価するため，センサー接続時からサーバの API で参照可能となるまでの時間を測定した．結果はセンサーの数を 80 個まで増加させても時間は一定であり，拡張性が実証できた．本研究の識別子を用いることで，センサーの遠隔管理方法の 1 つとして貢献できる．

1. はじめに

近年，IoT は発展し世界的な広がりを見せている [1]．IoT とは，物理的なオブジェクトがインターネットに接続されることによって，相互に情報交換を行う仕組みのことである [2]．Ericsson 社のレポートによれば 2020 年に世界で 12.6 億デバイス以上が存在し，2026 年には 26.9 億デバイスになると予想されている [3]．総務省の 2020 年度情報通信白書においても 30 億デバイス以上になると予想されており，世界的に増加していることがわかる．これは IoT が多くの分野に適用が可能であり，私達の身近な場所でもスマートホームやスマートシティという形で利用されているからである [4]．

1.1 背景

日本国内では，農業への取り組みを農林水産省が主体となって行っており，スマート農業実証プロジェクトをはじめとする実際の生産現場へのスマート農業の導入が顕著である．農林水産省によれば，根幹的農業従事者は 2020 年には 136 万人であり，2015 年の 176 万人と比べ減少傾向にある．さらに，依然として人手に頼る作業や熟練者でなければできない作業が多い．そこで，農業と IoT 技術を組み合わせたスマート農業によって作業の自動化や情報共有の簡易化を行い，これらの問題を解決している，しかし，導入や設置には学習コストが発生する点や，作物 1 つ 1 つや栽培区画の細かいチェックができない点が問題となっている．このようなスマート農業で用いられるセンサーは，IoT デバイスに組み込まれているものが多い．センサーが組み込まれている IoT デバイスは，NTT Communications 株式会社の Things Cloud や京セラコミュニケーションシステム株式会社の Sigfox が例として挙げられる．これらは，作物 1 つ 1 つの細かい範囲での監視に対して IoT デバイスを導入する場合，金銭的コストが高くなる．

本研究ではセンサーの数を自由に付け替え可能であり農業のユースケースに合わせた実験及び構築が可能な点から，Raspberry Pi や ESP32 に代表されるワンボードコンピュータとアナログ値を数値として出力可能なセンサーモジュールとして温度センサーや湿度センサーを対象とした．ここで，本研究での IoT デバイスとはワンボードコンピュータを指し，センサーとはワンボードコンピュータに接続されたセンサーモジュールのことを指す．これにより，IoT デバイス 1 つあたりに複数個のセンサーを取り付け，管理することが可能となる．

また，センサーやデバイスの個々の識別が可能になれば，数十万以上の大量のセンサーや生成された大量のデータの管理が実現できると Ciro Formisano らによって述べられている [5]．IoT 技術によるネットワーク内のオブジェクトは，一意に識別することによってオブジェクトの相互接続を可能にし，生成された大量のデータを管理する手法の 1 つとなり，これらが必要であると述べている．本研究では，ネットワーク内に存在する IoT デバイスに取り付けているセンサーをオブジェクトとして一意識別することによって，スマート農業におけるシステム設置のコスト削減及びスマートシティをはじめとする IoT の発展に貢献できる．

ここで，農業をはじめとする第一次産業に適応できる

[1] Tokyo University of Technology Graduate School Cloud and Distributed Systems Laboratory, Japan
[a)] g21210300d@edu.teu.ac.jp
[b)] kushida@acm.org

IoT デバイスとして，本研究では Raspberry Pi に焦点を当てている．これは，コンピュータボードの発展により，教育の観点だけでなく，前述の農業やヘルスケアをはじめとする産業に応用可能になっている [6]．Raspberry Pi とは，免許証ほどのサイズのシングルボードコンピュータであり，小型かつ軽量という特徴がある [7]．本研究では Raspberry Pi に標準搭載されており，入出力の両方に利用可能であり，有効・無効の設定も可能な点から GPIO 方式に焦点を当てている．なお，GPIO は General Purpose Input/Output の略であり，汎用入出力を意味している [8]．本研究によって, 前述の農業への適用のように，複数箇所に複数のセンサーを配置する環境下で遠隔監視を実現する手法の 1 つとして貢献可能である．

1.2 課題

本研究で解決する課題は，センサーには基本的に識別子がないため，一意に識別できないことである．世界中に普及している IoT デバイス及びセンサーは，個人あるいは組織単位で管理がされるが，組織において管理する台数は，数十万以上に及ぶため，個々の区別が必要となる．この IoT デバイスやセンサーへの ID 生成や割り当てにかかる時間は，台数が増えるにつれ増加する．また，管理する IoT デバイスやセンサーの数は定まった値ではないため，ID 割り当てや参照する時間が単調増加になると，単一の組織で管理可能な数は限られるという拡張性のない点が問題となる．既存の ID 割り当てに関する解決策として，IoT デバイスごとに 1 つのセンサーに限定する方法やセンサーを IoT デバイスに組み込むことでセンサーが識別情報を予め保持できるレジスタや機能を追加することである．どちらの考え方も単一のセンサーしか扱うことができない点や IoT デバイス単位での導入コストがかかる問題がある．

識別子を使用せずセンサーを識別する方法としては，機械学習があげられる [9]．センサーによって得られる実測値から分類を行うため，識別子が存在しない場合でも識別が可能である．しかし，各センサーの識別精度は，機械学習における学習量や学習モデルによって異なる．センサーを一意に識別するためには，正確な識別が不可欠である．センサーを一意に識別できない場合，あるセンサーを特定する際はユーザはその場で 1 つ 1 つ手探りをする必要がある．例えば，温室の温度を測定する際，あるセンサーからデータシートの誤差範囲を超えた値が出力された場合，どのセンサーかを特定し，原因の分析やセンサー交換を行う必要がある．センサーを特定するためには，1 つ 1 つを識別できる必要がある．I2C 方式が使用可能なセンサーは，I2C アドレスという識別番号を保持しているが，IoT デバイスの MAC アドレスのようにユーザが変更することが可能である．また，センサーは識別子として扱うことが可能なメーカー ID やシリアルナンバーを保持可能だが，これらの識別子を持つセンサーは一部に限られている．

2. 関連研究

IoT は農業での利活用の試みは既に行われている．Mahammad らはバッグボーンとしてクラウドコンピューティングを使用した農業 IoT のセンサー監視について調査した [10]. また, この論文においても一意の ID が必要になると言及されている．IP アドレスを識別子として扱うことが述べられているが，IP アドレスは変化する可能性があるため，一意に識別するためには固定化あるいは動的にセンサーと IP アドレスを紐付ける仕組みが必要となる．また，IPv6 の機能の 1 つであるインターフェース ID を用いる方法も存在するが，この ID 構造は MAC アドレスを拡張したものであり，MAC アドレスと ID の間に依存関係が発生してしまい，識別子として扱う場合は MAC アドレスが変化しない場合に限定する必要がある．さらにセンサーは MAC アドレスを保持していないため，センサーが接続する IoT デバイスの MAC アドレスを用いる，あるいは仮想的に付与する必要がある．

農業利用だけでなく，IoT を監視システムに利用した研究も存在する [6]．この研究では，Raspberry Pi と GSM モジュールを用いた制御と MySQL データベースを組み合わせ，モニタリングによって収集されたデータを医療関係に利活用できるとしている．ここでの GSM モジュールとは，GSM ワイヤレスネットワークで動作するワイヤレスモデムであり，シリアル通信や RS232 インタフェースを使用して Raspberry Pi や PC をはじめとするデバイスと接続することが可能である．実装として心電図の状態を監視し，心拍数が正常範囲内かどうかを判断し，閾値外の場合はブザーを鳴らした．しかし，心拍数をはじめとする医療データを読み取る際には部屋単位でしか判別をしておらず，センサー単位での詳細な識別が考慮されていない．

Jahoon Koo らは IoT プラットフォームで動作するデバイス ID の相互運用性に関する分析とデバイスネームシステム (DNS) アーキテクチャの提案を行った [11][12]．主に oneM3M, Oliot, Watson IoT, IoTivity, FIWARE の 5 つについて言及されている．これらの特徴を表 1 に示す．ここで，OID(Obejct IDentifier) とは，国際電気通信連合 (ITU) および ISO/IEC によって共同で標準化されている世界的に重複しない一意な識別子を割り当てられるために作られた識別子メカニズムである [13]．ASN.1(Abstract Syntax Notation One) で指定されたツリー構造を持つものを指している．oneM2M ではまず，higher arc として M2M Node Indication ID つまりは，OID 手順に従って定義および管理される．次に x, y, z, a とそれぞれ，メーカー ID，製品モデル ID，シリアル番号，そして拡張 ID を保持している．GS1 Oliot においても，OID，ID Keys，タイプで構成され，value 属性値とセットで呼び出される．IBM Watson

Vol.2021-DPS-187 No.31
Vol.2021-MBL-99 No.31
Vol.2021-ITS-85 No.31
2021/5/28

IoT では，クライアント ID ベースであり，Watson IoT Platform の組織 ID である orgid，デバイスのタイプ，デバイスの ID で構成されている．OCF IoTivity では，リソースのタイプやデバイスに割り当てられている ID をベースとしており，デバイス ID である”di”や，リソースタイプである”rt”と oic で決められた oic.d.[*] あるいは oic.wk.d が用いられている．なお，OIC とは Samsung，Intel，Cisco，MediaTek をはじめとする企業が主導する IoT 標準化団体である [14]．FIWARE では，一部の文字を除き，特定の制限がない形で構成されている．筆者たちは IoT DNS を実装した後，マイクロコンピュータでテストを行った．実験として oneM2M ベースのデバイスが Waston IoT および FIWARE センサーデバイスへのリソース要求を検証した．その結果，ユーザは異なる IoT プラットフォーム間で要求したリソースとサービスに合わせて適切に実行ができたと述べられている． この論文では各 IoT プラットフォームや識別子について触れているが，リソース要求形式の動的変換が主であり，識別 ID を統合する仕組みが必要不可欠となる [15][16].

表 1 既存識別子の比較

名前	識別子の構造
oneM2M	{(higher arc) (x) (y) (z) (a)}
GS1 Oliot	(GS1 OID).(ID Keys).(ID Key Type)
Watson IoT	_orgid:_devicetype:_deviceid
IoTivity	[di], oic.d.[*], rt:oic.wk.d
FIWARE	一部の文字を除き，特定の制限はなし

Huansheng らは IoT における非 ID オブジェクトのツリーコードモデリングとアドレス管理について定義と主な重要性について述べた [17]. しかしながら, 異種センシングデータの統合やアドレッシング時間の制御, リアルタイム性の課題が残っていると述べている．本研究ではプラットフォーム依存を解決するため, この論文で述べているモデリングやアドレス指定技術をさせることで, より汎用性の高さを維持しつつ数十万以上の IoT デバイスを管理可能な拡張性の高い IoT システムの実装が可能になる．

3. 提案

本研究では，識別子を持たないセンサーに対してソフトウェアを用いて新しい識別子を動的に割り当てることによって，課題の解決を行う．本研究の全体アーキテクチャを図 1 に示し，新たに割り当てられる識別子の構造とユーザによるセンサーの参照を図 2 に示す．

本研究の提案システムは図 1 のルートサーバ，API サーバ，IoT デバイス，センサーの 4 階層によって構成されている．ルートサーバは 1 台，それ以外は 1 台以上の構成である．ルートサーバ，API サーバ，IoT デバイスは HTTP 通信によってデータの送受信を行い，センサーは IoT デバイスに銅線が配線され，IoT デバイスはセンサーからソフトウェアを通じて実測値を取得している．

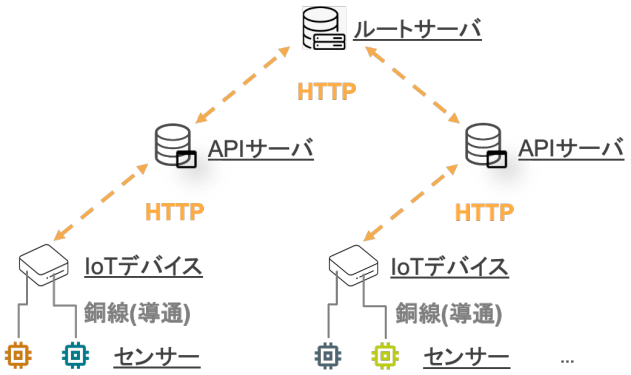

図 1 提案システムの全体アーキテクチャ

識別子は，ルートサーバが API サーバに割り当て管理を行うように親ノードが子ノードの識別子を管理する木構造を採用した．木構造の利点は，一対多の関係で構成され，各ノードは子ノードの識別子のみを管理するため，分散管理が可能な点である．これは管理している親ノードが変化した場合でもこれまでと同様に識別子に紐づいている情報を取り出すことが可能である．図 2 の識別子”1212030178” の IoT デバイスが識別子”1171780300”の API サーバから移動し，API サーバの識別子が変わった場合でも，API で参照する識別子はこれまでと同様である．

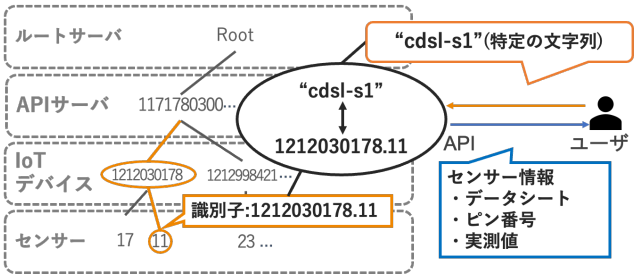

図 2 新たに割り当てる識別子構造とユーザによる参照

API サーバ及び IoT デバイスの識別子は，図 2 のように 10 桁で表現され，センサーの識別子はピンの番号を使用している．ユーザはセットアップ時にセンサーに”cdsl-s1” のような任意の文字列を割り当てることで，その名前で参照が可能である．識別子はノードが上位のノードに登録された日付と時刻をハッシュ化し，生成される．MAC アドレスや IP アドレスのように変化可能な値を使用しないため，生成された識別子はハードウェアやプラットフォームに依存せず，固定である．また，ハッシュを用いることで IoT デバイスや API サーバのデータベース内での衝突率を低下させることが可能である．識別子はルートサーバ及び API サーバ，IoT デバイスのデータベース内で図 3 の対応表によって管理される．ここで，API サーバに付与する識

Vol.2021-DPS-187 No.31
Vol.2021-MBL-99 No.31
Vol.2021-ITS-85 No.31
2021/5/28

別子を”サーバ ID”，IoT デバイスに付与する ID を”デバイス ID”，センサーに付与する ID を”センサー ID”として以降の文章では表記する．図 3 の矢印で示された値は，図 2 の識別子の例を表したものである．

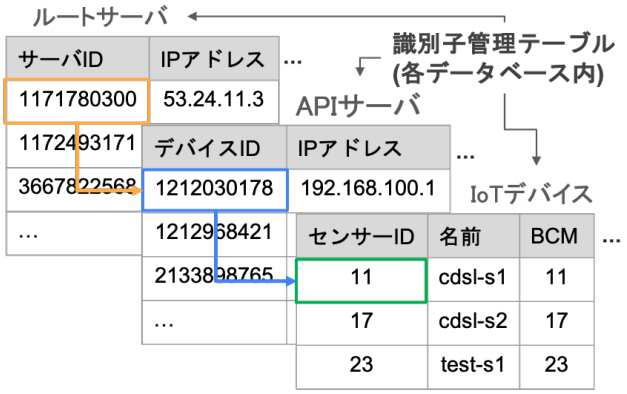

図 3　データベースを用いた識別子の対応表

API が通常ユーザによって識別子の検索を行う際は”デバイス ID. センサー ID”の形で参照される．図 2 の例では，ユーザが”cdsl-s1”という名前を指定し，対応表で名前が一致する各 ID を検索していき，”1212030178.11”が見つかるため，センサー情報の取得が可能となる．各 ID は各ノードに関連する値として IP アドレスやピン番号 (BCM) と紐づけられている．対応表は，センサーの取り付け及び取り外しによるイベントが発生するたびに更新されるため，高速な ID の紐付けや付け替えが可能である．センサーに紐付けられる名前は，IoT デバイスのセットアップ時にユーザによって対話形式で入力され，その値が設定される．これによって，ユーザは図 2 のように”cdsl-s1”とすれば対象となるセンサーを参照することができる．これら全体のソフトウェア構成を図 4 に示す．

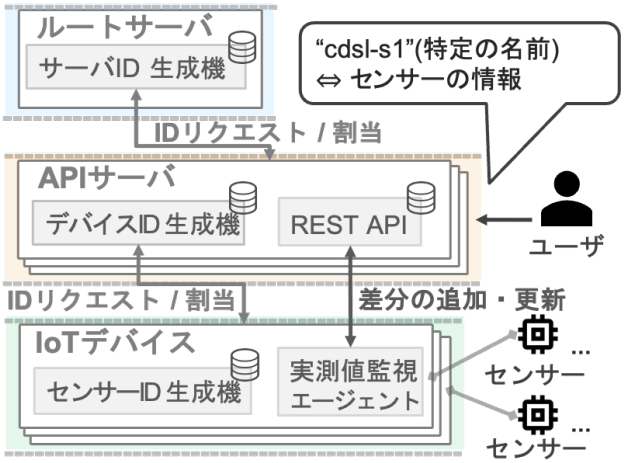

図 4　提案システムのソフトウェア構成

IoT デバイスのセンサー ID 生成機では，センサーが接続されているピン番号を取得し，センサー ID として登録し，ユーザによって決められた文字列の紐付けと管理を行う．センサーの導通確認と実測値の取得を定期実行及びセンサーの取り付け・取り外しによるイベント管理を行うものが実測値監視エージェントである．また，一定間隔ごとに実測値と ID の対応表の差分を API サーバのデータベースへ送信し，最新の状態を一定間隔で保つようにしている．一般的な IoT デバイスはストレージ容量が小さいことがほとんどであり，単一の IoT デバイスに多くのデータを集約することは分散管理ができない．そのため，ID の対応表と実測値のキューイングを行い，差分更新のみを行うことで保存するデータ量を削減している．

API サーバのデバイス ID 生成機では，デバイス ID の生成と IoT デバイスの MAC アドレスや IP アドレスと紐付けを行い，データベースでの管理を行う．また，ユーザがセンサーを参照するために REST API を配置している．API はユーザが定めた文字列をリクエストとして受け取り，デバイス ID 及びセンサー ID の変換を行い，データシートや実測値，ピン情報をユーザに返す．

ルートサーバのサーバ ID 生成機では，デバイス ID の生成方法と同様にしてサーバ ID を生成し，API サーバの IP アドレスと紐付けを行い，データベースでの管理を行う．サーバ ID は IoT デバイスの移動に伴い発生する部分木の位置変更を実現し，IoT デバイスの親ノードがどの API サーバでも付け替えが可能である．

本研究では，単純な木構造を拡張し，API サーバ間通信を実現することで IoT デバイスの移動を行わなくても他の API サーバ管理下の参照が可能となる．これらの用途について図 5 に示す．

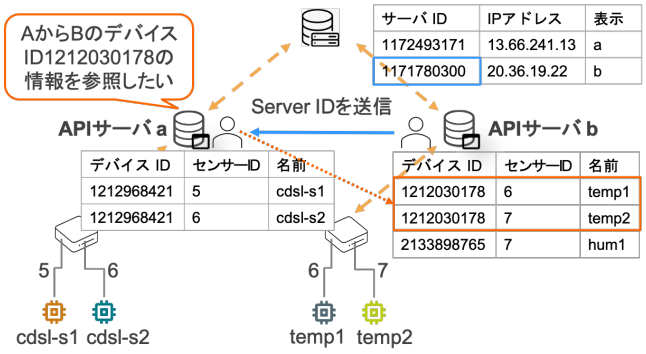

図 5　API サーバ間通信の用途

API サーバ a と b が存在し，各データベースは図 5 のようになっていたとする．ここで，API サーバ a のユーザが API サーバ b のデバイス ID”1212030178”の参照を可能としたい場合を例として挙げる．これは，他の組織で管理しているセンサーの情報を IoT デバイス単位で参照したい場合である．例として，他の支部や他の国で管理しているセンサーデータを機械学習データセットとして参照したい場

Vol.2021-DPS-187 No.31
Vol.2021-MBL-99 No.31
Vol.2021-ITS-85 No.31
2021/5/28

合である．API サーバの情報を共有したい相手に対して，サーバ ID を共有し転送設定を行うことで API サーバ a のユーザは API サーバ b の情報が参照可能となる．データ参照が可能になるまでの流れを図 6 に示す．

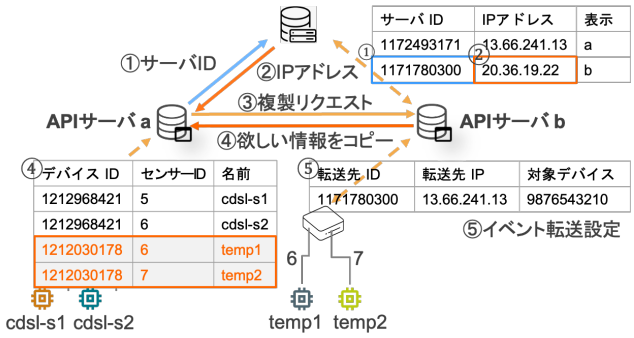

図 6　データ参照までの流れ

まず，API サーバ a からルートサーバへ IP アドレスの参照を行う．ルートサーバへサーバ ID を送ることで，サーバ ID に対応する IP アドレスを取得する．次に，API サーバ b にデータベースの複製リクエストを送信する．API サーバ b は対象となるデバイス ID に紐づけられた情報を API サーバ a にコピーを行う．最後に API サーバ b でイベント転送設定を行う．図 6 の 5 はルーティングテーブルの役割するため，一定間隔ごとに更新データを転送先に転送する．図 6 の例の場合，デバイス ID が”1171780300”に紐づけられた情報の更新がある場合は一定間隔ごとに API サーバ a に転送を行うことで，最新の情報を維持することが可能となる．一度キューイングを行うことで転送処理によるネットワークの輻輳を回避することが可能となる．

3.1　ユースケース

本研究の提案システムは，図 7 のように遠隔監視を実現するスマート農業への利用が可能である．

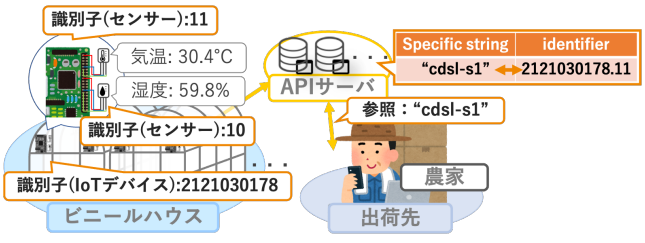

図 7　活用例 (ビニールハウス)

図 7 では，複数の位置にセンサーを配置しなければならない環境として，ビニールハウス内の作物を遠隔監視する例を表している．ユーザである農家が，ビニールハウスの各棟の四隅に配置されたセンサーから，実測値や活動の有無を遠隔地である出荷先から閲覧及び管理をしている．セットアップ時には業者あるいは農家本人がセンサーを設置し，センサーのピン番号の設定や任意の文字列による名前付けを行い，環境に関する実測値 (温度データや湿度データ) を収集する．図 2 で置き換えると，農家がユーザとなり，”cdsl-s1”という名前で 30.4 ℃を出力している温度センサーを参照したい例である．API サーバ上で”cdsl-s1”は 2121030178.11 に変換され，参照が可能である．

具体例として，ビニールハウス内でいちごをはじめとする農作物を栽培しているとする．ビニールハウスによる栽培は季節に関係なく作物に適した環境を作ることが可能だが，環境づくりを怠るとメリットがなくなる．その中でも特に温度管理は重要となる．いちごの葉の光合成の最適温度は 20〜23 ℃，適温域は 15.4 ℃〜27.4 ℃といわれている．夏場であればハウス内が高温になると同時に湿度も上昇し，病害虫が繁殖しやすい環境になってしまう可能性が高い．また，冬場であれば低温による作物の枯死の可能性がある．このように温度の管理は農業にとって必要不可欠であり，この管理をユーザにとってより容易でよりコストがかからないことが重要となる．本研究の提案システムを用いることで，農家はビニールハウスを常に傍らで見守る必要がなく，出荷先で遠隔監視することが可能となる．また，業者がセンサーを新規に取り付ける場合や交換した場合にも自動的に反映されるため，農家はビニールハウスで立ち会う必要がない．これは，センサーが破損し，異常値の出力やデータ送信が無くなった場合の特定にも有効である．

また，本研究で対象としているようなセンサーの場合，実測値となる入力信号以外に情報を保持していない場合が多い．その場合，生成されるデータは実測値やいつ取得したかのみである．そのため，タグ付けを行なったり，機械学習による分類を行ったりという方式も存在する [18]．しかしながら，そのデータはどのセンサーから生成されたものであるのか，データを生成したセンサーの規格をはじめとするデータシート情報はどういうものかがユーザからは把握ができない．センサーの詳細情報や IoT デバイスのどこに接続されているかという物理的な情報をユーザが把握することで，遠隔監視におけるセンサーの詳細を特定するまでのステップや時間を減少させることが可能である．また，この物理的な情報はセンサーが故障した，あるいは異常値を取得した場合にも活用が可能である．センサーの仕様として規格や型番，誤差範囲と物理的な接続位置が分かることでセンサーの管理者は迅速な原因究明が可能となる．

4.　実装と実験環境

図 2, 4 の実装について述べる．ルートサーバと API サーバは Ubuntu18.04 の Virtual Machine(VM) を使用し，IoT デバイスは Raspberry Pi 3 Model B+を使用し，センサーは DHT11 と ADT7410 をベースにソフトウェアによるエミュレートによって実現した．

提案システムの構築の流れを図 8 に示す．ルートサーバ

Vol.2021-DPS-187 No.31
Vol.2021-MBL-99 No.31
Vol.2021-ITS-85 No.31
2021/5/28

はユーザが関わるところではなく，全体を統括しているものであるため，ソフトウェア起動を前提としている．

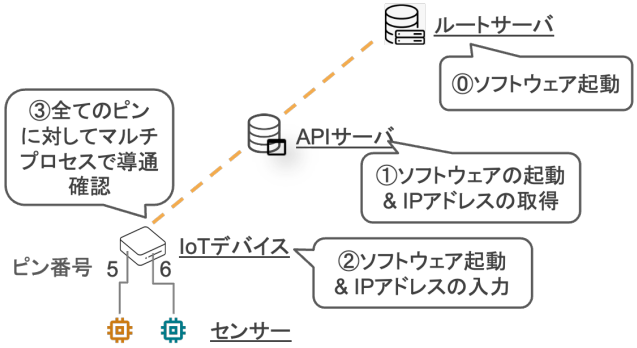

図 8 提案システム構築の流れ

まず，ユーザは API サーバで提案システムのソフトウェアの導入と起動を行う．データベースの構築と API によるデータ受信設定をシェルスクリプトによって自動的に環境構築が可能である．ユーザは起動のみすれば使用可能となるのである．次に，IoT デバイスもユーザがソフトウェアの導入と起動を行う．API サーバ同様にユーザがソフトウェア起動後，データベースの構築とライブラリの取得，定期実行の設定が自動的に行われる．その後，ユーザは参照される API サーバの IP アドレスを対話形式で入力する．これによって IoT デバイスのデータベース内で管理している識別子と実測値の API サーバへの差分更新が可能となる．更新頻度が多い場合，輻輳が発生する可能性があるため，初期設定では 1 時間ごとに送信を行っている．その後，毎秒導通確認用のソフトウェアが cron によって実行され，IoT デバイスの全てのピンに対して導通確認を行っている．これによってセンサーの取り付けや取り外しが 1(sec) 単位で検出することが可能となる．

本研究の各 ID は，親ノードに登録した日付と時刻をハッシュ化することで生成している．生成の流れを図 9 に示す．サーバ ID 及びデバイス ID は Cyclic Redundancy Check(CRC) を用いて日付をハッシュ化し，これによって ID 同士の衝突の確率を低下させている．CRC は巡回冗長検査と呼ばれ，誤り検出符号の一種であるが，生成される CRC 値はメッセージとの 1 対 1 の対応が不可能というハッシュ関数と同等の性質を持ちつつ，値が小さいという特徴がある [19]．ハッシュ化する方法は CRC 以外にも存在するが，ソフトウェアのコーディングに使用した Python3 で容易に実装でき，10 進数での出力という点で検査値の長さが 32 ビットになる CRC-32 を採用した．日付情報のフォーマットは”%Y-%m-%d %H:%M:%S:%f”であり，例として”2021-04-19 23:59:59:000001”のような年月日と時分秒に加え，マイクロ秒まで取得している．これは，同時多接続の場合に被ってしまうケースを少しでも軽減させるためである．もし，衝突した場合は再度生成を行う．

セットアップ時およびデータ更新時には毎回現在接続されているセンサーの状態を確認する必要がある．その際には WiringPi ライブラリの gpio readall コマンドによって現在 Mode が IN となっているピン番号 (BCM) を取得し，出力結果の整形を行った上でピン 1 つ 1 つに入力信号の有無を確認する．初期セットアップ時に Mode が IN かつ 3.3V や 5V，GND を無くした場合に残るのが 26 個のピンとなる．これらに対して導通確認や識別子の割り当て，実測値の取得を行う．センサーに対する識別子の割り当てでは，生成された識別子が衝突しないかを判断する．イメージ図を図 9 に示す．図の左側はデータベース内のテーブルの動きを表し，右側では疑似コードを表している．

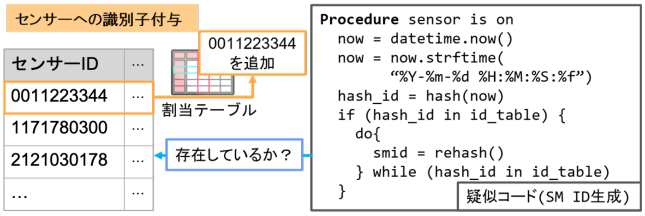

図 9 サーバ ID 及びデバイス ID 生成

実験はセンサーが IoT デバイスのピンに接続されてから API から参照可能になるまでの時間を計測している．API サーバ上のデータベースにデータがアップロードされるまでの時間は，センサーの数が増加した場合にどの程度増加するのか定量評価を行う．センサーの数が増加した場合でもセットアップ時間がほとんど変わらない場合，提案システムはセンサーの数の拡張性を実証可能である．

5. 実験結果

システムを評価するためにセットアップ時間を計測した．ユーザがセンサーを接続し，セットアップ完了後に測定値とステータスを API サーバに送信し，反映されるまでの時間を計測した．全てのピンに対して線形にチェックを行った場合のセットアップ時間の計測結果を図 10 に示す．

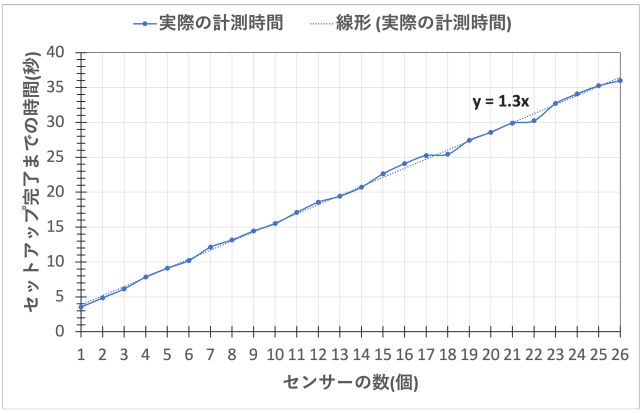

図 10 基礎実験によるセットアップ時間の計測結果

Vol.2021-DPS-187 No.31
Vol.2021-MBL-99 No.31
Vol.2021-ITS-85 No.31
2021/5/28

エンドツーエンドモデルによって 1 つ 1 つ追加していく場合のケースである．実験はセンサーの数を増加して測定を行った．セットアップ時間を 100 回計測し，その平均値を図 10 に示している．セットアップ時間は一連の処理を自動化したが，線形増加する結果となった．

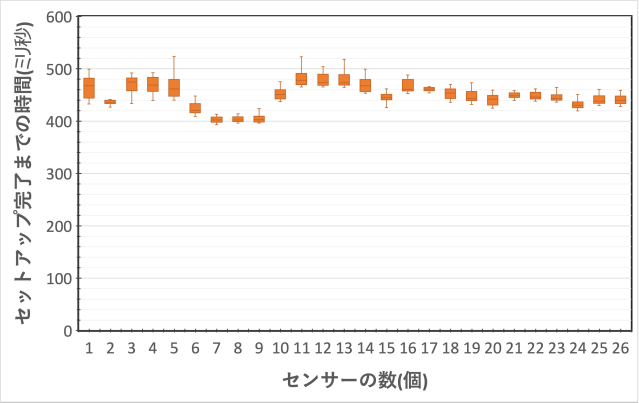

図 11 Raspberry Pi 想定でのセットアップ時間の計測結果

センサーをソフトウェアで制御し，マルチプロセス化と自動化処理の追加を行った結果を図 11 に示す．図 10 と同様にセットアップ時間を 50 回計測し，箱ヒゲ図にした．これは，Raspberry Pi での使用を基準として，物理的に接続できるピンの数が 26 であったため，センサーの数を 26 個まで増加させた．図 11 では，セットアップ時間はほとんど 400(mec)〜500(msec) の範囲にあり，ほとんど変化していないことがわかる．また，平均値から生成された近似直線の傾きは 10^{-4} 以下であり，変化が小さいことがわかる．

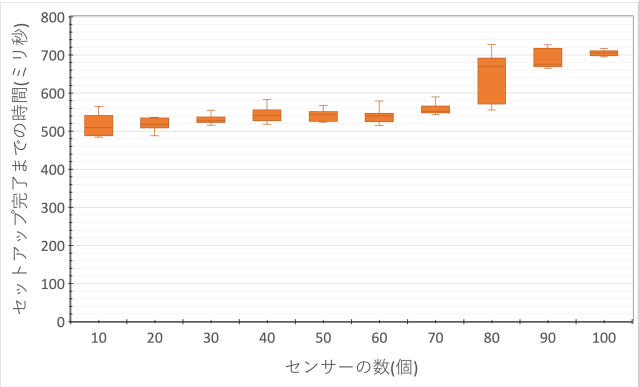

図 12 論理上接続可能な数でのセットアップ時間の計測結果

センサーの数を論理上可能な数に増やして拡張実験を行った．実験結果を図 12 に示す．センサーの数が 80 個の時点でセットアップ時間が急激に増加した．センサーの数が 80 個以上になると直線的に増加し続けた．

6. 評価と分析

マルチプロセス化と自動化による時間の短縮により、セットアップ時間の一定化を行った．セットアップ時間の 400(mec)〜500(msec) の内訳は，ピン情報の取得・実測値の確認・DB への格納・サーバーへの送信処理の 5 つである．100(mesc) 以上時間が経過した処理はピン情報の取得とサーバーの処理の 2 つであった．

ピン情報の取得に 100(msec) 以上時間が経過する理由は，今回の実験で使用した GPIO 方式を用いるセンサーは，実測値取得のために開始信号を送ってから待ち時間が発生したためである．センサーによって待ち時間が異なるため，本研究では登録されているセンサーの中で最も待ち時間が長いものを採用した．登録されてるセンサー全てに対応ができる反面，待ち時間が必要以上に長くなるセンサーもあるため，この待ち時間もデータベースで管理し，個々のセンサーに合わせて待ち時間を設定することで時間の削減が可能である．

サーバーの処理時間に 100(msec) 以上時間が経過した理由は，マルチスレッドではない点及びネットワークのオーバーヘッドである．サーバー処理をマルチスレッド化し，同一ネットワークやホップ数が小さいサーバーを用いることで時間の短縮が可能である．

7. 議論

本研究の提案におけるセンサー ID の生成および付与のやり方では高速化という観点では，セットアップ時間の減少量に限りがあり，改善する必要性がある．これは，センサーの導通確認やデータ取得に GPIO 方式を用いているため，IoT デバイスから開始信号をセンサーに送信し，実測値が返ってくるまでに数十 ms〜数百 ms の一定の待ち時間が発生するためである．この時間はセンサーごとに異なるため，登録されている全てのセンサーから実測値を必ず取得するためには，登録されているセンサーの中で GPIO 方式かつ最も時間がかかるセンサーの待ち時間が経過する必要がある．本研究の実験時では DHT11 により，最低でも 180(msec) の待ち時間が発生していた．待ち時間の発生しない I2C(Inter-Integrated Circuit) や SPI(Serial Peripheral Interface) 接続方式に変更することでよりセットアップにかかる時間を削減可能である．また，導通確認時は実測値の取得も同時に行っていたため，待ち時間が発生していた．センサーの読み取り専用の即時取得可能な値でチェックをすることでより時間を削減可能である．

実験時では，VM と Rasoberry Pi を用いたため，API サーバの台数あるいは IoT デバイスの台数を 1 万台以上用意することが非現実的であった．そのため，IoT デバイス及びセンサーをソフトウェアエミュレートし，API サーバもコンテナ化することでより 1 万台以上を想定とした検証

が可能になる．

また，現在のライブラリの取得方法はリアルタイムでの追加ができないため，新規のセンサーが開発された場合対応ができない．これはセンサーの種類の識別の際に，既存のライブラリのラッパーを作成し使用しているため，ライブラリが存在しない場合，センサーの特定が不可能となってしまう．ライブラリの検索方法においても線形探索しているため，検索時間は登録されているライブラリの数に依存する．この検索方法も同様に高速化が必要であり，実測値を集計し分類し，可能な限り一意に近い形での特定が必要となる．

8. おわりに

本研究では，センサーに対して新たな識別子を割り当てることで，識別情報を保持していないセンサーを個々に識別することが可能となった．提案システムの実装後，識別子が割り当てられ，実際に実測値を取得するまでの時間を計測し，センサーの数が増加した場合の時間の変化について実験を行なった．IoT デバイスに接続されているセンサーの数が増加した場合でも，単一の IoT デバイスが保持しているピンの上限数までは時間の変化は 100(msec) 以下であった．この結果から，提案システムはセンサーの数の増加が時間の増加に影響を受けず，拡張性がある．そのため，組織が 1 つ以上で運用されている環境下での IoT デバイス及びセンサーの遠隔管理に貢献可能である．

謝辞

本研究は，JSPS 科研費 JP20K11776 の助成を受けたものである．

参考文献

[1] Novo, O.: Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT, *IEEE Internet of Things Journal*, Vol. 5, No. 2, pp. 1184–1195 (online), DOI: 10.1109/JIOT.2018.2812239 (2018).

[2] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, *IEEE Communications Surveys Tutorials*, Vol. 17, No. 4, pp. 2347–2376 (online), DOI: 10.1109/COMST.2015.2444095 (2015).

[3] Bugel, J., John, S. and Schwartz, S.: Ericsson Mobility Report, Technical report, Ericsson (2020).

[4] Jin, J., Gubbi, J., Marusic, S. and Palaniswami, M.: An Information Framework for Creating a Smart City Through Internet of Things, *IEEE Internet of Things Journal*, Vol. 1, No. 2, pp. 112–121 (online), DOI: 10.1109/JIOT.2013.2296516 (2014).

[5] Formisano, C., Pavia, D., Gurgen, L., Yonezawa, T., Galache, J. A., Doguchi, K. and Matranga, I.: The Advantages of IoT and Cloud Applied to Smart Cities, *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 325–332 (online), DOI: 10.1109/FiCloud.2015.85 (2015).

[6] Gupta, M. S. D., Patchava, V. and Menezes, V.: Healthcare based on IoT using Raspberry Pi, *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 796–799 (online), DOI: 10.1109/ICGCIoT.2015.7380571 (2015).

[7] Severance, C.: Eben Upton: Raspberry Pi, *Computer*, Vol. 46, No. 10, pp. 14–16 (online), DOI: 10.1109/MC.2013.349 (2013).

[8] Vujović, V. and Maksimović, M.: Raspberry Pi as a Wireless Sensor node: Performances and constraints, *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1013–1018 (online), DOI: 10.1109/MIPRO.2014.6859717 (2014).

[9] Shafique, M., Theocharides, T., Bouganis, C., Hanif, M. A., Khalid, F., Hafiz, R. and Rehman, S.: An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era, *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 827–832 (online), DOI: 10.23919/DATE.2018.8342120 (2018).

[10] Mekala, M. S. and Viswanathan, P.: A Survey: Smart agriculture IoT with cloud computing, *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, pp. 1–7 (online), DOI: 10.1109/ICMDCS.2017.8211551 (2017).

[11] Koo, J. and Kim, Y.: Interoperability of device identification in heterogeneous IoT platforms, *2017 13th International Computer Engineering Conference (ICENCO)*, pp. 26–29 (online), DOI: 10.1109/ICENCO.2017.8289757 (2017).

[12] Koo, J., Oh, S.-R. and Kim, Y.-G.: Device Identification Interoperability in Heterogeneous IoT Platforms, *Sensors*, Vol. 19, No. 6 (online), DOI: 10.3390/s19061433 (2019).

[13] Dubuisson, O.: *Introduction to Object Identifiers (OID) and Registration Authorities*.

[14] Park, S.: OCF: A New Open IoT Consortium, *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 356–359 (online), DOI: 10.1109/WAINA.2017.86 (2017).

[15] Da, B., Esnault, P. P., Hu, S. and Wang, C.: Identity/identifier-enabled networks (IDEAS) for Internet of Things (IoT), *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 412–415 (online), DOI: 10.1109/WF-IoT.2018.8355102 (2018).

[16] Wu, C., Lin, F. J., Wang, C. and Chang, N.: OneM2M-based IoT protocol integration, *2017 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 252–257 (online), DOI: 10.1109/CSCN.2017.8088630 (2017).

[17] Ning, H., Yang Fu, S. H. and Liu, H.: Tree-Code modeling and addressing for non-ID physical objects in the Internet of Things, *Telecommunication Systems* (2014).

[18] Mahdavinejad, M. S., Rezvan, M., Barekatain, M., Adibi, P., Barnaghi, P. and Sheth, A. P.: Machine learning for internet of things data analysis: a survey, *Digital Communications and Networks*, Vol. 4, No. 3, pp. 161 – 175 (online), DOI: https://doi.org/10.1016/j.dcan.2017.10.002 (2018).

[19] Peterson, W. W. and Brown, D. T.: Cyclic Codes for Error Detection, *Proceedings of the IRE*, Vol. 49, No. 1, pp. 228–235 (online), DOI: 10.1109/JRPROC.1961.287814 (1961).