木構造を用いた識別子の動的な割当による センサーモジュールの一意識別

高木 優希[1,a)]　串田 高幸[1]

概要：近年，IoT の発展と同時にその管理が注目を集めている．特にデータを生成するセンサーの管理には物理的位置の把握とそのデータシート情報が必要となる．しかし，電気信号のみを出力するような低レベルのセンサーは識別情報を持たないことが多いため，ネットワーク上で識別が不可能となる．また，新たに識別子を付与した場合にソフトウェアに依存することが多く，問題となる．さらに，膨大な量のセンサーを管理する場合に管理コストが高くなるという問題点もある．これらの問題点を解決するため，木構造を用いた動的な識別子の付与を提案する．日付情報をハッシュ化したものを新規 ID として付与することで依存関係の解消と管理の容易さを向上させた．また，既存方式よりもユーザー負荷を減少させることによって管理コストの低下につなげた．これらを従来の手法と比較し，セットアップまでの時間的コストの低下とユーザー負荷の減少を目的としている．本研究によって IoT 学習環境の整備を向上させ，IoT 利用可能分野の拡張を実現する．

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]．Ericsson 社のレポートによれば 2019 年には世界で 10 億デバイス以上が存在し，2025 年には 24.6 億デバイスになると予想されている [2]．総務省の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 [3]．この膨大な数の IoT デバイスを管理する方法を確立させることで，日々進化している IoT の技術に貢献できると考えられる．日本国内においても IoT は発展しており，産業やコンシューマ向けに今後も増加することが予想されている．これらは単一領域内では完結せず，相互に関係しあうことで社会全体への効果が期待される．そこで，本研究の成果であるネットワーク内におけるセンサーの一意識別を可能とすることで IoT の利用分野を拡張することを実現する．

背景

IoT は多くの分野に適用が可能なものであるが，前述したようにデバイス数が今後も増加することを考慮すると，IoT デバイスから生成される膨大なデータの管理が重要となる．Ciro Formisano らは解決方法の 1 つとして, IoT 技術によるネットワーク内オブジェクトの一意識別を挙げた [4]．本研究では特に取り付け型のセンサーモジュールに焦点を当て，一意識別を実現する．この研究によって, 複数箇所に複数のセンサーを配置する環境下で真価を発揮することが可能となる．

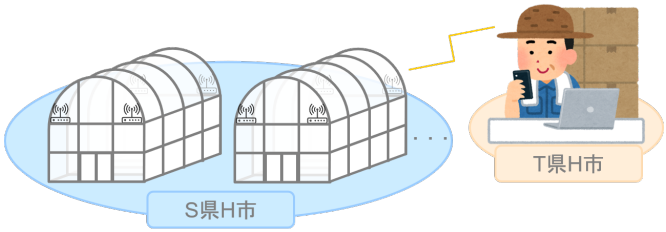


図 1　使用例 (ビニールハウス)

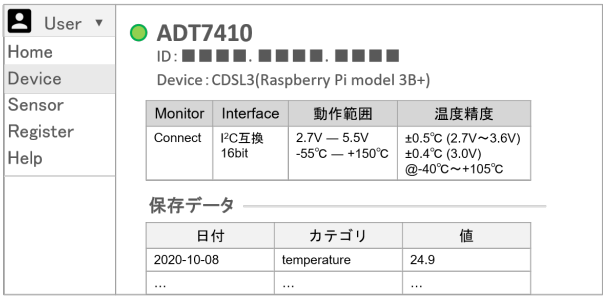


図 2　想定図

図 1 ではビニルハウスの各棟にセンサーを配置し，温湿

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[a)] C0117178

度をはじめとする室内の情報を取得する例を示している．ユーザーは図 2 のようなダッシュボードから IoT デバイスの登録や取得データを確認できることを想定している．通常，Web のダッシュボードを介してセンサーを管理することが一般的ではあるが，既存のものでは対処しきれない点が 2 つある．その問題点について図 3 で説明を行う．

図 3　ダッシュボード管理による想定と問題点

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，実測値やいつ取得したものかというデータしか生成されない．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．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といった点がユーザーからは把握ができない．センサーの詳細情報や位置といった物理的な情報をユーザーが把握することで，遠隔監視の管理コストが低下すると考えられる．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．規格や型番，誤差範囲といったセンサーの仕様と物理的な位置情報が分かることでユーザーは迅速な原因究明が可能となる．

課題

本研究で扱う課題は以下の 3 点である．

- 全てのセンサーモジュールが識別情報を保持しているわけではない点
- 識別子がソフトウェアやプラットフォームに依存する可能性が高い点
- 組織単位では管理コストが大きい点

1 つ目の課題としては，ネットワーク上で全てのノードを識別するためには，オブジェクトが必ず一意に識別できなければならない [4][5][6][7]．既存方式では，オブジェクト識別子やクライアント ID を付与する，あるいはリソースの形式を機械学習による分類やデバイス ID 付与することで解決している．本研究においても新たに一意の識別子を付与している．2 つ目の課題としては，新たに付与した識別子や元から保持している識別情報の多くが提供元企業や組織のデバイスやプラットフォームに依存することが多い [5]．また，乱数や IP アドレスといった変動が起こりうるものを拡張する形では，オブジェクトが再接続された場合に対応ができなかったり，依存関係が発生してしまうため他の環境で使用不可になる場合があったりと問題がある．識別子の管理とデータの管理を区別するような形を取ることで依存関係が解消されるため，本研究でもこれを採用している．3 つ目の課題としては，ユーザーによる一元管理の場合，図 4 の様に組織の再編や変更は物理的な移動や手動での再登録作業が伴うため，ユーザーへの負荷が増加するという問題が発生する．図 4 あるいは Web サービスのダッシュボードを用いる場合も考えられるが，企業が提供するものや OSS どちらにおいても問題点がある．前者であれば特定の製品化されている者しか対応が不十分であったり，企業の提供するプラットフォーム以外で識別子が使えななかったりといった問題がある．後者であればエンドツーエンドがほとんどであり，取り付けを行うようなセンサーは対応ができないという問題がある．

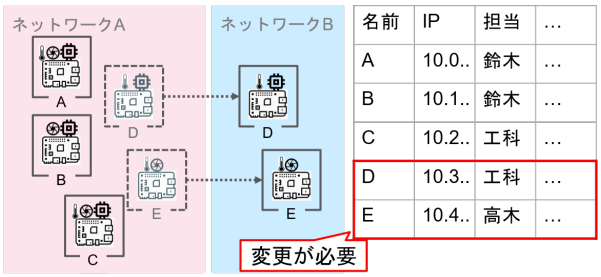


図 4　ユーザーによる管理の例

各章の概要

2 章では本研究と関わりのある研究について取り上げ，それぞれの特徴と問題点を述べる．3 章では本研究の提案内容について具体的に触れ，アーキテクチャの説明を詳細に行う．4 章では具体的な実装方法を詳細に説明するとともに実験環境について述べる．5 章では 4 章の実験結果を元に評価および分析を行う．6 章ではこれまでの提案および評価について議論を行う．最後に 7 章ではこれまで説明した課題，提案，結果をまとめる．

2.　関連研究

この章では本研究と関連した先行研究について取り上げ，その関連性について述べる．

図 1 のように，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．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 [7]．また，IPv6 プロトコルを用いた方法も考えられているが，IPv6 の機能の 1 つであるインターフェース ID は MAC アドレスを拡張したものであり，依存関係の解消にはならない．

識別子の付与やノードの一意識別は先行研究においても課題として挙げられている．Jahoon 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を行った [5]．オブジェクト識別子をベースとする oneM2M および GS1 Oliot やクライアント ID ベースの IBM Watson IoT, リソースの形式やデバイス ID をベースとする OCF IoTivity について, フォーマットやメタデータが言及されている．ここでのオブジェクト識別子とは, Abstract Syntax Notation One(ASN.1) で指定されたツリー構造を持つものを指す．本研究で扱うものに関してもオブジェクト識別子ベースに類似するため, oneM2M および GS1 Oliot との比較も評価で行う. この論文で述べられている様に, プラットフォームに依存した識別子も存在するため, 識別 ID を統合する仕組みも必要となってくると考えられる．

さらに Jahoon Koo らは翌年に上記の研究をより深め，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 [6]．主に oneM3M, Oliot, Watson IoT, IoTivity, FIWARE の 5 つについて言及されている．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．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，依存関係の解消にはつながっていない．

次章で詳しい説明を行うが, 本研究では識別子の管理として木構造モデルを採用している．Huansheng らは本研究と同様の 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 [8]．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, リアルタイム性といった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とさせることで, より汎用性の高さを維持しつつ識別可能上限数の多い IoT システムの実装が可能になると考えられる．

3. 提案

この章では，本研究の提案内容について詳細に述べる．本研究は, 各ノードに対して新規に生成した識別子を付与し, それらを木構造として各上位ノードで管理することで 1 章で述べた 3 つの課題を解決する．1 つ目の課題であるすべてのセンサーが識別情報を保持していない点は，新たに木構造の識別子を付与することで解決を行う．本研究で生成した識別子の構造を図 5 に示す．まず，root は組織用の ID を自動的に割りあてる機能を持ち，1 台〜数台を想定している．これは現在の DNS のルートサーバーと同様の役割を想定しているからである．次に，bunch では各 grain へ ID の割当を行う．最後に，grain では各 Sensor Module(SM) へ ID の割当を行う．このように階層構造にすることで，機器の物理的な移動が発生した際に部分木として取り出し，再構築することを可能とするためである．これらの識別子はソフトウェアとは切り離し，API とすることで 2 つ目の課題であるソフトウェアやプラットフォームとの依存関係の解消を行う．具体例を図 6 に示す．

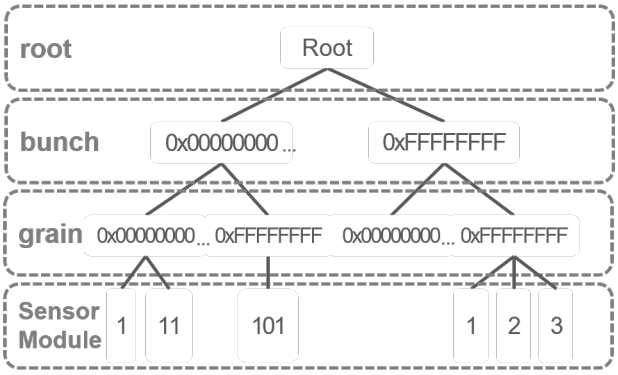


図 5　識別子とノードの関係図

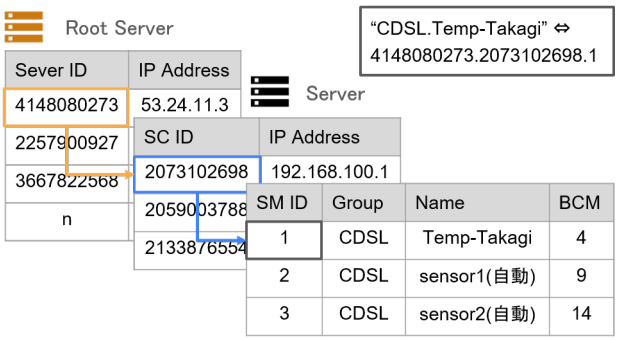


図 6　ID のテーブル例

ユーザーがセンサーに対して”Temp-takagi”のようなユニークな名前と所属グループを決定し，その名前と ID を対応付けている．グループは変更可能な名前空間であり，物理的な移動が発生した場合でもグループが変更されることはない．1 度に大量のセンサーを接続した場合，ユーザーの名づけが困難になる．それを解消するため，名づけを行わない場合は自動で名前が付けることで解決している．ユーザーはこの名前を Web のダッシュボードか grain で確認することが可能である．また，3 つ目の課題である管理コストであるが，ユーザーがセットアップ時に触れなければならない数を最低限削減することでユーザーへの負荷を減少させる．これを実現するために，grain と bunch 間でそれぞれクライアントソフトウェアおよびサーバーソフトウェアの実行を行い，接続可能な状態を構築する．つまり，root - bunch 間，SM - grain 間，bunch - grain 間といった順序で実行を行う．この ID 生成時の構成を図 7 に

示す．bunch や root ではアクセス時の日付データを用いて一意の識別子を生成する．

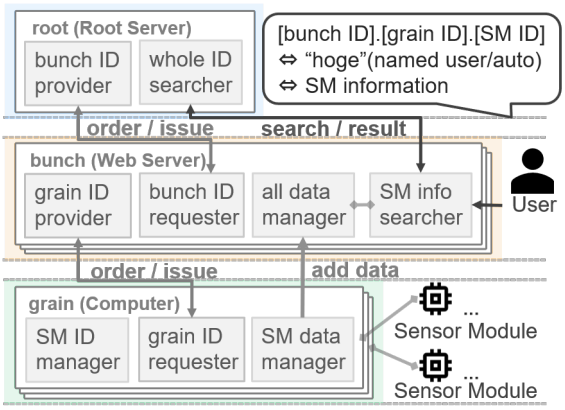


図 7　全体構成図

図 6 に示すように，10 桁の 10 進数で表現され，復号すれば ID 生成時の日付情報も把握が可能である．grain では SM の ID を管理するが，SM の入力信号の有無や BCM, Mode といった判断材料を元に接続状態を把握する．接続状態にあるセンサーに対し，順に ID を割り振っていく．I2C 規格の ID を 10 進数変換することで用いることも可能である．ID 生成を行うための環境構築の具体的な挙動について図 8 に示す．

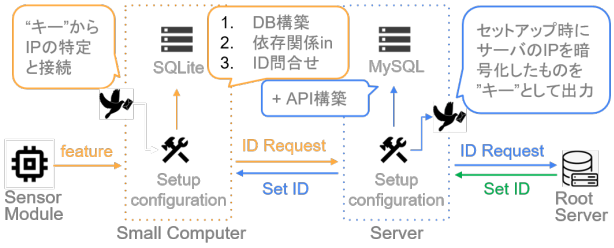


図 8　環境構築時の挙動

図 8 では, クライアントソフトウェアおよびサーバーソフトウェアを実行した後の動作を示している．データベースの構築，依存関係のインストールを行った後，サーバーの IP アドレスをキーとして grain 側で入力することで ID のリクエストおよびその後の通信ができるようにする．サーバー側では ID 問い合わせまで行った後，API の構築を行い，ユーザーが外部から ID を扱える環境構築を行う．また，SM の情報を取得するために，ピン番号 (BCM) および Mode，データ信号の有無を確認した後，呼び出すことが可能なライブラリによってどのセンサーであるかを判断している．ソフトウェアが動作後，ID 検索時について図 9, **??**に示す．図 9 では，grain から bunch へデータベースの更新を一定間隔で行う．通常ユーザーは WebUI から，開発者や管理者であれば WebUI か裏側で動作している API を用いることでデータベースの値を取得することが可能である．具体的な API におけるリクエストとレスポンスに関して図 10 に示す．

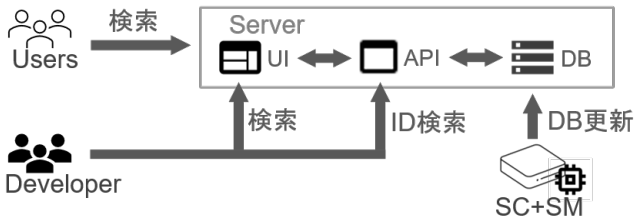
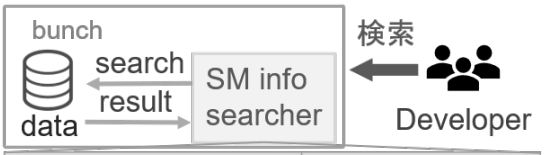


図 9　ID 検索時 (通常)



APIへ渡す値	戻り値
デバイス名(string)	ID
グループ名.デバイス名	ID
bunchID.grainID.SMID	センサー情報一覧
ID.sheet	データシート情報
ID.data	実測値
ID.where	センサーの位置
ID.memo	ユーザーメモ

図 10　API のリクエストとレスポンス

ユーザーはセンサーの情報を取得したい場合，そのセンサーの ID を把握しなければならない．そのため，ユーザーが名づけを行った場合はデバイス名，同時多数の登録のために名づけを自動化した場合はグループ名. デバイス名を API に渡すことで ID が取得可能である．この ID に加えて取得したい情報をドット区切りで渡すことでそれぞれ登録されている値が戻り値となる．見つからなかった場合は戻り値は空である．

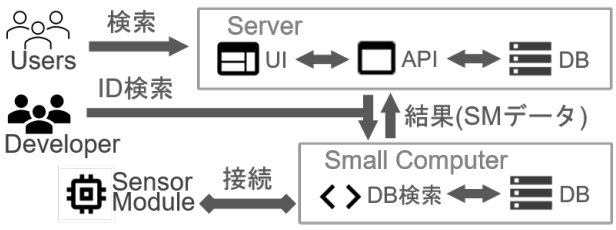


図 11　ID 検索時 (ダイレクト)

図 11 では，bunch と grain が同一ネットワーク内のみ使用可能な特殊な検索方法である．現在の稼働状況や実測値をリアルタイムで取得したい場合に使用する．直接 grain

側の DB にアクセスし，最新の情報をリクエスト時に取得する．

4. 実装と実験環境

この章では，提案のソフトウェアの実装と実験環境について述べる．実装では，クライアントソフトウェアおよびサーバーソフトウェアの設計とその方法を具体的に記述する．実験環境では，実装を行ったソフトウェアの開発環境と実験する際の環境や構成について記述する．

4.1 実装

ID 生成時のフローを図 12 に示す．図 12 では，SM として 3 つの異なるセンサー (ADT7410, DHT11, HC-SR501)，grain として Raspberry Pi，bunch として Ubuntu 上に nginx を導入し動作の検証を行った．

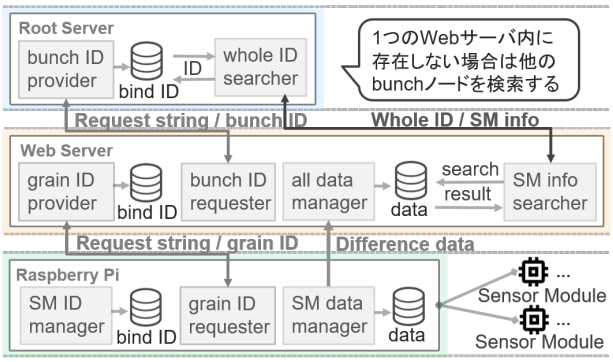


図 12　ソフトウェア構成図

図 12 では，図 7 の具体的な実装である．bunch 側で ID を生成する際 CRC32 を用いて日付をハッシュ化している．これによって ID の衝突の確率を低下させている．grain 側では，DB 内で空いている ID を線形探索によって取得し，I2C の ID が使用可能かどうか，閾値を超えていないかといったチェックの後に割り当て処理を行う．このデータ更新は 10 分毎に行われており，ユーザーがセンサーを抜き差しした場合や追加した場合をこのデータ更新前にチェックする．一定間隔で自動的にセンサーの状態と ID の状態を更新可能である．grain では識別子だけでなく，実測値を一時的に格納している．grain 上で管理する実測値のデータは 1 時間ごとに bunch へ送信し，ID 情報と共に差分の同期を行うようにしている．これによってキューの役割を果たし，トラフィックの増加を低下させる．これらの環境構築はシェルスクリプトを bash で起動し，bunch からのキーを入力すれば自動的にすべてのセットアップができるようにしている．シェルスクリプト内では，依存関係のインストールおよび実行環境のアップデートを行っている．図 8 の bunch からのキーは入力後，設定ファイルへリダイレクトし，宛先を常に把握するようにしている．ID 生成に関する具体的な流れを図 13，14 に示す．図 13 では，ID 問い合わせの内容について示している．grain は bunch に，bunch は root にそれぞれ上位のノードに対して ID 問い合わせのリクエストを送信する．grain と bunch 間は図 8 によって接続ができ，root に対しては固有の IP が決まっているため，元々サーバーソフトウェアに記述している．POST 形式でリクエストという意味の Type が 1 と，送信元の IP アドレスおよび MAC アドレスを送信する．これによって下位ノードから上位ノードだけでなく，上位ノードから下位ノードへのアクセスが可能になり，相互通信を可能としている．

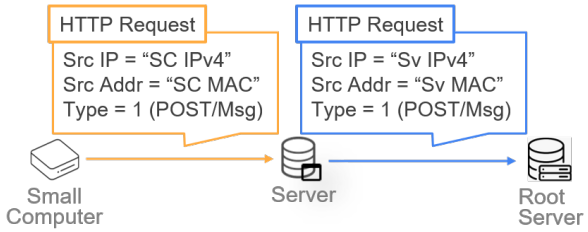


図 13　ID 問い合わせのリクエスト

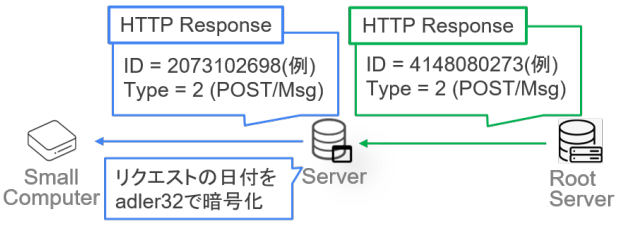


図 14　ID 問い合わせのレスポンス

図 14 では，ID 問い合わせの後の動作を示している．HTTP Request があった日時をハッシュによって暗号化し ID としている．この生成された ID とレスポンスの意味の Type=2 を返す．ハッシュを用いるため一意性が保たれ，ID が衝突する可能性は低くなると考えている．SM に対する ID は整数値を順に割り当てる方式をしている．線形探索によって空いている ID を順に割り当てている．ハードウェア上 grain が接続できる SM の数が限られているという点からこの形式を採用した．これらの一連の流れによって ID をそれぞれ取得および割り当てを行う．ユーザーは図 2 の様に Web からアクセスを行うことで ID に紐づけられたセンサーから取得した実測値を確認できる．また，ダッシュボードの裏側で動作している API にアクセスできるようにし，URI あるいはパスにアクセスすることで値が取得できる REST API を想定している．

4.2 実験環境

実験を行った環境について表 1 に示す．本研究ではフリーの Linux ディストリビューションの 1 つである Ubuntu を主に使用した．データベースはリレーショナルデータベースを使用し，データ構造にルール性を持たせた．ソフトウェアは主に Python3 によって記述し，その他シェルスクリプトによって動作している．サーバー側では Flask を用いることで python を動作させている．実験に使用した構成を図 15 に示す．

表 1　実験に使用した機器

表記名	使用 OS/使用機材
Root Server(root)	Ubuntu20.04LTS + MySQL5.7
Web Server(bunch)	Ubuntu18.04LTS + MySQL5.7
Computer(grain)	Raspbian4.19 + SQLite3.33
Sensor Module	ADT7410/DHT11/BMP180/HC-SR501

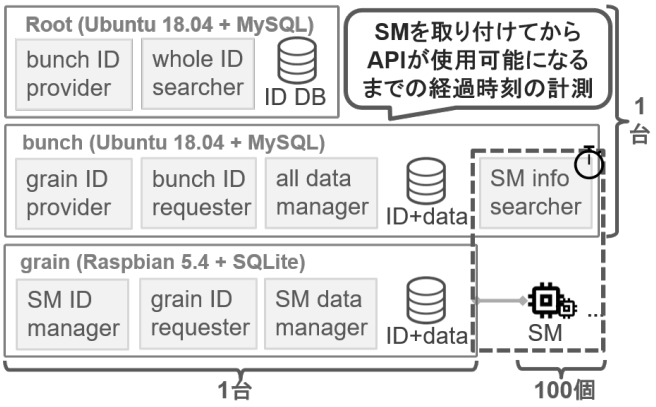


図 15　実験の構成図 (データ取得時間)

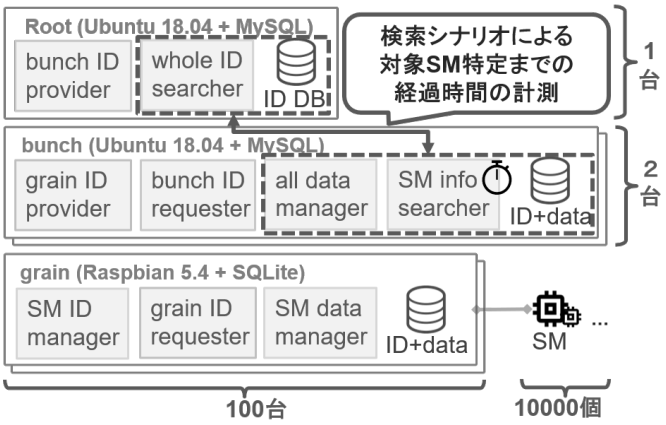


図 16　実験の構成図 (検索時間)

実験はセンサーの数を 1〜100 個まで増加して行う．これは単一の grain で管理できる数の上限が 200 前後であることから，まず grain を 1 台センサを可変にして実験を行う．次に，grain の台数を増加するとともに，SM を 1000 台まで増加して実験を行う．bunch 側と grain 側でそれぞれソフトウェアを実行しなければならないため，フェーズを分けて時間を計測する．このセットアップまでの時間と API から検索を行い，結果が返されるまでの時間を既存の OSS と比較する．SM の台数に関わらず高速なセットアップおよび検索を検証する．

5. 評価と分析

この章では実験に対する評価および分析の方法について述べる．評価項目として以下の 3 点を実施予定である．

(1) 従来の方式と導入からデータ取得までの時間計測 (x 軸をセンサー数，y 軸を時間とする)
(2) 従来の方式と検索時間の計測 (x 軸をセンサー数，y 軸を時間とする)
(3) IPv6 のインターフェース ID や PIM プロトコルのインターフェース ID をはじめとする標準化されている識別子との比較

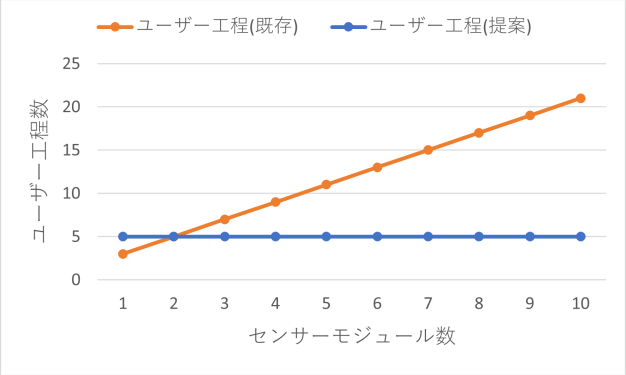


図 17　既存方式と提案方式のユーザー負荷の推定

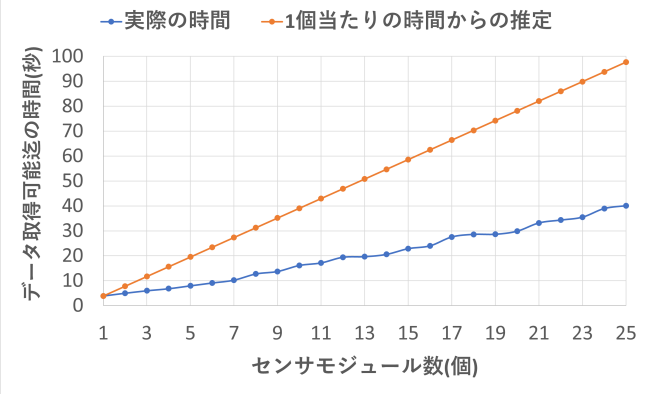


図 18　SM 数の増加によるデータ取得までの時間推移

一般的な IoT 管理ソフトウェアでは初期セットアップ後，センサーモジュール数が増加するほどクライアントおよびサーバー側のソフトウェアに変更あるいは修正が必要となる．そのため，セットアップ項目を 1, クライアント

およびサーバーのソフトウェアへの変更・修正をそれぞれ 1 ずつとすると図 17 のようなグラフになると予想される．提案方式ではセットアップ項目が既存方式より多い反面，その後は自動的に反映される形式を用いているため，ユーザーが関わる工程を削減でき，セットアップまでの時間を短縮できると考えられる．これによって課題の 1 つである管理コストの低下を解決できると考えられる．まず，図 15 を検証した．図 18 では横軸をセンサーモジュール数，縦軸を経過時間としてグラフ化したものである．センサー 1 個当たりにかかる時間を 100 回計測し，その平均値を基準とした際，その時間× n 個の予測値が橙色の線である．これは，既存方式によくある形式であり，1 つ 1 つのセンサーに対してセットアップを行う形である．それに対し，提案方式では既存方式を大きく下回り，緩やかに増加していることが見て取れる．センサーの数が増加した場合でも既存方式より高速な識別とデータ取得が可能になる．図 16 に関しては今後の活動とする．

6. 議論

本研究の提案では機器の IP アドレスや MAC アドレスに依存しない形で ID を付与しているため，課題であった依存関係の解消は実現できたと考えられる．しかしながら，IP アドレスが分からない場合，データの送受信が不可能なため，データベースによる対応表を用いた管理は依然として必須である．この IP アドレスと提案した識別子を動的に紐づけることで透過的にしている．ここで，現在採用さている識別子との比較を行う．一般的に世界的規模で扱うことのできる UUID や GUID では一意性は大きく保障されている [9]．さらに，ユーザー個人で生成可能という点からソフトウェアやプラットフォームに依存しない形式である．しかし，本研究のような木構造ではないため，データの生成元を追跡する場合に困難となる．つまり，一意性が強すぎることによって，ネットワーク外のノードから値を取得する場合に特定が困難になってしまうということである．これらも本研究と同様に IP アドレスと共に用いることでノードの識別は可能である．しかし，世界中のノードは全てが 1 つの組織で管理されているわけではなく，組織が集合したものである．つまり，1 つの組織で扱える数には限界があるため，本研究のように柔軟に変更が可能な形式にすることが管理コストの低下につながるのである．また，IPv6 で使用されるインターフェース ID というものも存在している [10]．しかしながら，これは MAC アドレスを拡張したものであり，依存関係を解消するためには本研究のような形式が必要不可欠である．本研究で実装した状態では不完全であり，外部から API を使用する形式が成立していないため，今後の課題である．また，複数の bunch 間での検索シナリオが root を介するため，これを柔軟に行う必要がある．その点を解消し，ユースケースを明確化することでより管理のしやすさが向上し，更なる課題解決につながると考えられる．

7. おわりに

最後に, 本レポートのまとめを述べる．本研究では, 3 つの課題を解決するべく研究を行った．1 つ目は, 識別情報を持つノードと識別情報を持たないノードが存在し, これらの一括管理を行うことが難しいということである．2 つ目は, 識別情報を付与する際に, アプリケーションやプラットフォームに依存しやすいという点である．3 つ目は，組織単位で管理を行う際に管理コストが高くなるという点である．これらの課題を解決するために木構造を用いた識別子を新たに付与することで，依存関係の解消とユーザー負荷の低下，つまり管理コストの低下を実現した．自動的な環境構築や動的な識別子の配置によって既存方式と比較して，ユーザーが触れなければならない工程を減少させ，ほとんどが工程の間にユーザーを介さず，ソフトウェアのみで一貫して処理が終了するよう実装を行った．また，識別子では IPv6 のインターフェース ID や UUID をはじめとする多くのものが標準化されているが，本研究で取り上げた 3 つの課題を全て解決しているのは提案方式だけである．しかしながら，本研究にはユーザーインターフェースやユースケースに対するアプローチが未完成であるため，今後のタスクとしている．本研究によって, IoT 管理が可能な分野の範囲が格段に広がると同時に, 発展と増加を続ける膨大な数の IoT デバイスの管理に貢献できるのである．

参考文献

[1] Novo, O.: Blockchain Meets IoT : A Architecture for Scalable Access Management in IoT, *IEEE Internet of Things Journal*, Vol. 5, No. 2, pp. 1184–1195 (2018).
[2] Hemmer, H., Ashraf, C. and Powell, A.: Ericsson Mobility Report, Technical report, Ericsson (2020).
[3] of Internal Affairs, M. and Communications: *White Paper on Information and Communications in Japan* (2020).
[4] 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* (2015).
[5] Koo, J. and Kim, Y.-G.: Interoperability of device identification in heterogeneous IoT platforms, *2017 13th International Computer Engineering Conference (ICENCO)* (2017).
[6] Koo, J., Oh, S.-R. and Kim, Y.-G.: The 6th International Symposium on Sensor Science, *Interoperable Device Identification System for IoT Sensors* (2018).
[7] Mekala, M. and Viswanathan, P.: A Survey : Smart agriculture IoT with cloud computing, *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)* (2017).
[8] Ning, H., Fu, Y., Hu, S. and Liu, H.: Tree-Code modeling and addressing for non-ID physical objects in the Internet of Things, *Telecommunication Systems* (2014).

[9] Leach, P., Mealling, M. and Salz, R.: *A Universally Unique IDentifier (UUID) URN Namespace* (2005).
[10] Carpenter, B. and Jiang, S.: *Significance of IPv6 Interface Identifiers* (2014).

