一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

状態遷移図ベースのコード生成とログ可視化機能を備えたセンサネットワーク実機検証基盤の開発

辻村篤志† 小林侑生† 不破泰† アサノデービッド†

† 信州大学 〒380-0928 長野県長野市若里 4-17-1
E-mail: †24w6047a@shinshu-u.ac.jp

あらまし センサネットワークの開発は幅広い専門知識と多大な工数を要する。先行研究では、センサネットワークの動作を表す状態遷移図からコードを生成し、汎用ハード上で動作する環境が構築されてきた。本研究ではその環境を拡張し、その上で実用的な無線通信プロトコルを実装・検証した。その過程で汎用ハード上でのプロトコルの動作が確認しづらいことを課題として認識し、デバッグ用ログ出力とそれを可視化・ステップ実行できる環境を開発しデバッグ効率の向上を図った。

キーワード 無線センサーネットワーク、無線通信プロトコル、状態遷移図、コード生成、ログ可視化

Atsushi TSUJIMURA†, Yu KOBAYASHI†, Yasushi FUWA†, and David ASANO†

† Shinshu University, 4–17–1 Wakasato, Nagano-shi, Nagano 380–8553 Japan
E-mail: †24w6047a@shinshu-u.ac.jp

Abstract The development of sensor networks requires extensive expertise and significant effort. Previous research has generated code from state transition diagrams representing the behavior of sensor networks, establishing an environment that operates on general-purpose hardware. This study expands that environment and implements and verifies practical wireless communication protocols. During this process, we recognized the difficulty in confirming the operation of protocols on general-purpose hardware as a challenge and developed an environment for debugging log output and visualizing and step-executing it to improve debugging efficiency.

Key words wireless sensor networks, wireless communication protocols, state transition diagrams, code generation, log visualization

1. はじめに

1.1 背　景

近年、無線センサネットワーク（Wireless Sensor Network：WSN）は、環境モニタリングやスマートシティ、インフラ監視、農業、ヘルスケアなど、さまざまな分野での応用が期待されている。実際、WSN を含む IoT デバイスの数は 2020 年時点で約 130 億台に達し、年率 20 %で増加しており、2025 年には 300 億台弱に到達すると予測されている [1]. また、MDPI の報告によれば、WSN ノードの数は 2012 年の約 4.5 億台から 2022 年には約 20 億台へと拡大しており [2]、その研究および実用の両面での発展が顕著である。しかし、これらのシステムを構築するためには、複雑な通信プロトコル設計やデータ処理アルゴリズムの実装が求められ、開発者には高度な専門知識と多大な工数が必要となる。そのため、システム開発の効率化や開発者支援を目的とした研究が進められている。

1.2 従来研究

従来研究では、無線通信プロトコルの状態遷移図から自動生成したプログラムを汎用ハード上で動作させ、CSMA および TDMA ベースの MAC プロトコルを実機で実装・検証できる環境が構築されていた（旭ら）。このシステムにより基本的な通信プロトコルの動作検証が可能となったが、デバッグ方法はシリアルコンソールへのログ出力に依存しており、通信処理が高速に進むため逐次的な状態遷移を追跡することが難しかった。その結果、異常動作の原因を把握しづらく、開発効率や教育的利用の観点では十分でない点が課題として残されていた。

1.3 目　的

本研究の目的は、従来研究で課題となっていたデバッグ効率の低さを改善し、実機でのプロトコル検証をより効果的に行える環境を実現することである。そのために、状態遷移図から生成したコードの動作をログとして出力し、それを可視化・ステップ実行できる仕組みを開発した。これにより、プロトコル

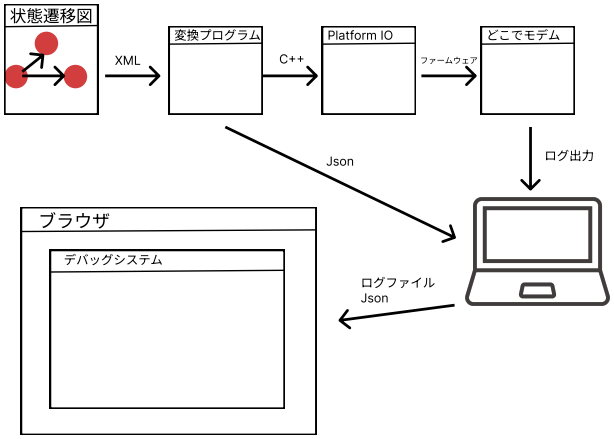

図 1　提案システムの全体構成
Fig. 1　Block diagram of the proposed system.

の動作を逐次的に把握でき、異常動作の原因究明を容易にするとともに、教育や応用実証に適した支援環境を提供することを目指す。

2. 提案システム

2.1 システム構成

本研究で開発したシステムの全体構成を図 1 に示す。本システムは、状態遷移図を入力として C++ プログラムを自動生成し、PlatformIO 上でビルドを行い、汎用ハードウェア上で実行することでセンサネットワークのプロトコルの動作検証を可能とする。さらに、通信動作の状態遷移や変数値の過程をログとして記録し、ブラウザ上で状態遷移の可視化およびステップ実行を行える環境を備えている。これにより、プロトコル設計から実機検証、デバッグまでを一貫して支援することが可能となる。

2.2 ファームウェア生成部

ファームウェア生成部では，無線通信プロトコルの動作を Astah Professional 上の状態遷移図として設計し，XML 形式で出力する。次に，変換プログラム（Python）を用いて状態遷移図の XML を C++ プログラムへ変換し，PlatformIO 環境でビルドすることでファームウェアを生成する。このファームウェアは 2.3 節で述べる汎用ハードウェア上で実行される。

先行研究（小林ら [4]）では，状態遷移図の各要素（通信状態・遷移・初期値設定・処理内容など）を XML ファイルから抽出し，それぞれをリスト化したうえで，通信状態間の関係を解析し，C++ の ‘switch-case‘構造に変換するアルゴリズムが実装されている。本研究におけるファームウェア生成部は，この変換処理を利用しており，状態遷移図で設計されたプロトコル仕様を汎用ハードウェア上で動作するプログラムとして自動生成するものである。

図 2 に状態遷移図の例を，プログラム 1 に変換後の C++ コードを示す。状態遷移図で定義された「待機（LISTEN）」「受信（RECEIVE）」「送信（TRANSMIT）」の各状態と遷移関係が，

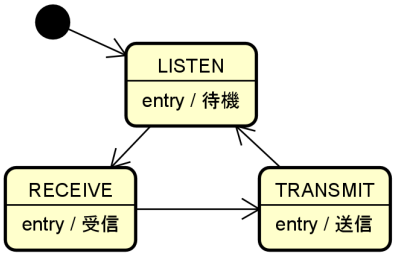

図 2　状態遷移図の例
Fig. 2　State transition diagram.

プログラム 1　変換された C++ プログラム

```
1  typedef enum { LISTEN, RECEIVE, TRANSMIT } NodeState
       ;
2
3  void protocol_main() {
4    switch (state) {
5      case LISTEN: updateState(RECEIVE); break;
6      case RECEIVE: updateState(TRANSMIT); break;
7      case TRANSMIT: updateState(LISTEN); break;
8    }
9  }
```

‘switch-case‘構造によって対応づけられており，設計段階の論理構造がソースコードとして正しく反映されていることが確認できる。

2.3 実行環境 (ハードウェア)

次に，実行環境について述べる。本システムでは，SAMD21 マイコンを搭載した無線モデム（株式会社サーキットデザイン製どこでモデム）を用い，生成したファームウェアを書き込み実行する。このモデムは 429 MHz 帯で動作する汎用的な無線モデムであり，技術基準適合証明を取得しているため，実際に電波を送受信しながら通信プロトコルの検証を行うことができる。また，429 MHz 帯は中山間地域において回折性能に優れており，低消費電力で長距離通信が可能な LPWA（Low Power Wide Area）通信に適している。この特性を活かすことで，将来的には登山者見守りシステムなどへの応用も期待できる。さらに，本モデムはマイコンと通信モジュールが一体化しているため，外部配線や追加ハードウェアを必要とせず，開発や実験環境の構築を容易に行うことが可能である。加えて，GPS モジュールを併用することで，位置情報の取得および 1PPS 信号による高精度な時刻同期を実現し，複数ノード間での無線通信プロトコルの正確な動作検証を可能としている。

2.4 デバッグシステム部

本節では，通信プロトコルの動作を可視化し，逐次的なデバッグを可能とするデバッグシステムについて述べる。提案システムは，ファームウェアが出力するログを入力として，ブラウザ上で状態遷移の再現と変数追跡を行う仕組みを備える。こ

図 3　どこでモデム (左) と GPS モジュール (右)
Fig. 3　Block diagram of the proposed system.

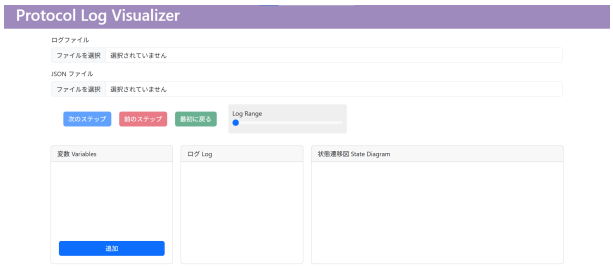

図 4　ログ可視化・ステップ実行環境の UI
Fig. 4　User interface of the log visualization and step execution environment.

れにより，開発者は通信の挙動を視覚的に理解でき，異常箇所の切り分けを効率化できる。

2.4.1　UI　構　成

図 8 に可視化画面の構成を示す。画面上部にログファイル (テキスト) および状態遷移図ファイル（JSON）のアップロード領域を配置し，中央にステップ実行用のボタン群（次，前，最初に戻る）とスライダを設ける。下部は「変数追跡」「ログ表示」「状態遷移図描画」の 3 領域で構成し，処理の流れと内部状態を統合的に把握できる設計とした。

2.4.2　ログ出力形式

各状態遷移および変数変化は，識別可能な最小単位で逐次ログ化する。状態は

`STATE:LISTEN`
`STATE:RECEIVE`

のように 1 行 1 状態で出力し，変数は

`[VAR:current_slot=3]`
`[VAR:has_own_slot=true]`

のように `[VAR:<name>=<value>]` 形式で出力する。値は数値・文字列の双方を許容し，可視化側で時系列に統合して解析する。単純で曖昧さの少ない表記とすることで，正規表現に基づく軽量なパーサで高速に処理できる。

さらに，ログ出力の形式にばらつきが生じないよう，C++ のマクロを用いてログ出力関数を定義している。このマクロは，変数名と値を自動的に取得して `[VAR:<name>=<value>]` 形式で出力するため，開発者は変数名を明示する必要がない。例えば，ユーザは `PRINT_VAR_LOG(変数)` のように記述するだけで統一フォーマットのログを生成できる。この仕組みにより，全ての出力が一定形式で記録され，可視化システム側で確実に解析可能となる。

プログラム 2　状態遷移図を表す JSON ファイルの例

```
{
  "states": [
    "LISTEN",
    "RECEIVE",
    "TRANSMIT",
    "WAIT_ACK"
    ],
  "transitions": [
    {"from": "LISTEN", "to": "RECEIVE"},
    {"from": "RECEIVE", "to": "TRANSMIT"},
    {"from": "TRANSMIT", "to": "WAIT_ACK"},
    {"from": "WAIT_ACK", "to": "LISTEN"}
    ]
}
```

2.4.3　可視化・再現機構

本システムでは，記録されたログファイルと，状態遷移図の構造を記述した JSON ファイルをもとに状態遷移を再現する。具体的には，JSON ファイルから状態名および遷移関係を抽出し，それらを Mermaid.js のフォーマットへ変換して描画処理に渡す。ログの進行に合わせて該当ノードをハイライトすることで，通信処理の時系列的な動作を視覚的に確認できる。

Mermaid.js は，テキストベースでフローチャートや状態遷移図を描画できる軽量な JavaScript ライブラリであり，HTML 上での埋め込みや動的描画に適している。本研究では，ブラウザ環境のみで動作し，追加ソフトを必要としない点，および構文が簡潔で自動生成との親和性が高い点から採用した。これにより，状態遷移図を表す JSON データから自動的に Mermaid 記法を生成し，`stateDiagram-v2` 形式で描画することで，設計段階の論理構造をそのままの形で可視化できる。

なお，可視化に用いる JSON ファイルは，2.2 節で述べたファームウェア生成部と同一の Python プログラムによって自動的に生成される。Astah Professional から出力された状態遷移図の XML を解析し，C++ コード生成と同時に，状態遷移関係を JSON 形式で出力している。この仕組みにより，設計情報と可視化データの不整合を防ぎ，設計・実装・可視化の一貫性を確保している。

図**??**に生成される JSON ファイルの例を示す。この JSON は，状態リストと遷移リストの 2 つの要素で構成されており，状態名およびそれらを結ぶ遷移関係を保持する。

ここで，`states` 配列は通信状態の一覧を表し，`transitions` 配列は状態間の遷移関係を示す。`from` および `to` がそれぞれ遷移元と遷移先の状態を表している。このデータ構造をもとに

プログラム 3　生成された Mermaid.js フォーマットの例

```
stateDiagram-v2
[*] --> LISTEN
LISTEN --> RECEIVE
  RECEIVE --> TRANSMIT
  TRANSMIT --> WAIT_ACK
  WAIT_ACK --> LISTEN
```

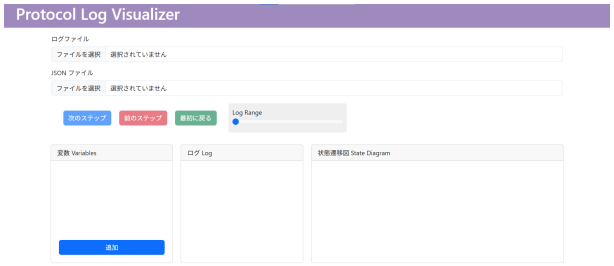

図 5　ステップ実行時の状態ハイライト例

Fig. 5　Example of state highlighting during step execution.

Mermaid.js 形式のテキストを自動生成し，ブラウザ上で状態遷移図を描画している。

変換後の Mermaid.js フォーマットをプログラム n に示す。

このように，シンプルな JSON 構造を中間表現として用いることで，XML から生成された設計情報を軽量に扱うことができ，状態遷移構造の可視化を容易に実現している。

2.4.4　ステップ実行および変数追跡

本システムでは，通信プロトコルの動作を 1 ステップ（1 状態遷移）単位で再現できるステップ実行機能を備えている。画面上部に配置された「次のステップ」「前のステップ」「最初に戻る」ボタンにより，状態遷移を順方向または逆方向に逐次移動でき，通信処理の流れを段階的に確認することが可能である。さらに，スライダを用いることで任意のステップ位置へ直接移動でき，興味のある箇所を即座に調査できる柔軟な操作性を実現している。

状態遷移の再生にあたっては，ログの各行を 1 状態単位として扱い，時系列に沿って状態情報を更新する。ログ再生に合わせて，中央のログ表示領域，右側の状態遷移図，および左側の変数表示領域が同期して更新されるよう設計されている。これにより，ある状態における変数値の変化や，その状態から次への遷移を同時に確認できる。特に，状態遷移図上では現在の状態ノードがハイライト表示され，通信動作の進行位置を視覚的に把握しやすい構成となっている。

変数追跡機能では，ログ出力された任意の変数を選択し，その値の変化を時系列に追跡できる。追跡対象の変数は複数追加可能であり，状態変化とあわせて観察することで，通信プロトコルの内部状態を多角的に解析できる。これらの機能により，特定条件下での変数の変化や分岐挙動を直感的に検証でき，デバッグおよび設計検証の効率向上に寄与する。

2.4.5　リアルタイム可視化

本システムの拡張として，実機で動作するファームウェアのログをリアルタイムに取得・可視化する機能を試作した。図**??**にその構成を示す。

デバイス上で動作するファームウェアは，通信動作に応じて状態遷移ログをシリアルポートへ逐次出力する。Python スクリプトがこのシリアルポートを常時監視し，取得したログデータを MQTT（Message Queuing Telemetry Transport）プロトコルを介してバックエンドサーバへ送信する。MQTT は軽量な Pub/Sub 型通信方式であり，組込みデバイスとサーバ間のリアルタイム通信に適している。

バックエンドでは，Node.js 上で動作するサーバが MQTT ブローカーからログデータを受信し，Socket.IO を用いてブラウザと双方向通信を行う。これにより，取得された状態遷移情報はフロントエンドへ逐次転送され，ブラウザ上の Mermaid.js による状態遷移図に即時反映される。通信遅延は数百ミリ秒程度に抑えられ，状態の変化をほぼリアルタイムに確認できることを確認した。

この構成により，従来の「ログ取得→解析→可視化」といった事後的なデバッグ手法に対し，動作中のシステム挙動を即時に観測できる「実時間デバッグ」が可能となった。これにより，通信プロトコルの動作確認やノード間同期の評価を効率化できるだけでなく，将来的には実験中の異常検知や教育用デモ環境への応用も期待される。

一方で，状態遷移の周期が非常に短いプロトコルや高速に状態が変化するシステムにおいては，描画更新が処理速度に追従できず，視認性が低下する場合がある。このため，本機能は主に周期の長い通信プロトコルや教育目的の可視化において有効であり，高速プロトコルの解析にはログ記録・再生型の可視化機構（2.4 節）を併用することが望ましい。

3.　プロトコル実装と評価

3.1　プロトコルの実装

状態遷移図をもとに自動生成されたファームウェアを実機に適用し、独自プロトコルを実装した内容を説明。書くべき内容：使用した状態遷移図の概要（図 7）各状態の役割（IDLE, TRANSMIT, WAIT_ACK など）コード生成後に PlatformIO で実行可能になるまでの流れ（図 1〜図 7 とつなぐ）どのようにスロット割当・送信・ACK 処理を行うか（簡略な説明）図 7 は簡略化している旨を明記（→論理構造に焦点）

例文イメージ：状態遷移図から自動生成されたプログラムをどこでモデム上で実行し、信大独自プロトコルを実装した。図 7 に本プロトコルの主要な状態遷移を示す。本図は、実装上の例外処理や再送制御を省略し、主要な動作経路を示している。

3.2　実　　験

実際に複数ノードで通信を行い、ログを取得した環境と条件。

書くべき内容：

実験環境（図 1 の汎用ハードウェア構成を引用）

送受信ノードの台数・配置（例：2 ノード間通信で検証）

検証項目：「通信状態の追跡」「デバッグの効率」「ログ可視化の有用性」

例文イメージ：

実験では，送信ノードと受信ノードの 2 台構成で通信プロトコルを動作させた。各ノードは SAMD21 マイコンを搭載した無線モデム上で動作し，状態遷移ごとにログを出力した。これらのログをブラウザ上に読み込み，状態遷移図と変数を逐次可視化することで，プロトコルの動作確認および異常時の原因追跡を行った。

3.3 評　　価

「従来との比較」「有効性の実証」「教育的有用性」を軸に。

書くべき内容：

コンソール出力のみとの比較（表 or 図）

可視化ツール使用時の違い（例：デバッグ時間短縮・理解のしやすさ）

図 8 として「ログビューワ画面」を掲載（実際の UI キャプチャ）

ステップ実行中の状態変化を図 9 として掲載

例文イメージ：

図 8 に提案システムの可視化画面を示す。ログファイルをアップロードすると，通信プロトコルの状態遷移がブラウザ上で描画される。図 9 にステップ実行中の画面を示す。現在の状態がハイライトされ，変数の値が逐次更新されることで，通信処理の挙動を直感的に把握できる。従来はコンソールログを手動で追う必要があったが，本システムにより動作過程を一目で確認でき，デバッグ時間が大幅に短縮された。

本システムの有効性を確認するため，状態遷移図から自動生成したコードを用いて無線通信プロトコルを実装し，複数端末間での通信を行った。その際に出力されるログをブラウザに取り込み可視化することで，プロトコルの状態遷移を逐次的に追跡できることを確認した。

従来はコンソール出力のみに依存していたため，高速に進行する通信処理の挙動を逐一把握することは困難であった。本システムではログを視覚的に再現し，ステップ実行形式で確認できるため，デバッグ効率の向上に寄与することを示した。

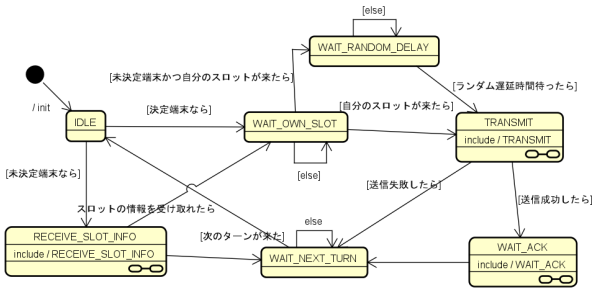

図 6　開発した無線通信プロトコルの状態遷移図．※実装上の詳細条件や内部処理は簡略化して記載している．

Fig. 6　State transition diagram of the developed wireless communication protocol.

```
STATE:IDLE
current_slot = 0
STATE:RECEIVE_SLOT_INFO
STATE:CHECK_RECEIVED_PACKET
STATE:WAIT_OWN_SLOT
current_slot = 1
STATE:WAIT_OWN_SLOT
STATE:WAIT_OWN_SLOT
current_slot = 2
STATE:WAIT_RANDOM_DELAY
STATE:CREATE_PACKET
STATE:TRANSMIT
・
・
・
```

図 7　従来のコンソールによるデバッグ方法

Fig. 7　Example of the old console-based debugging method.

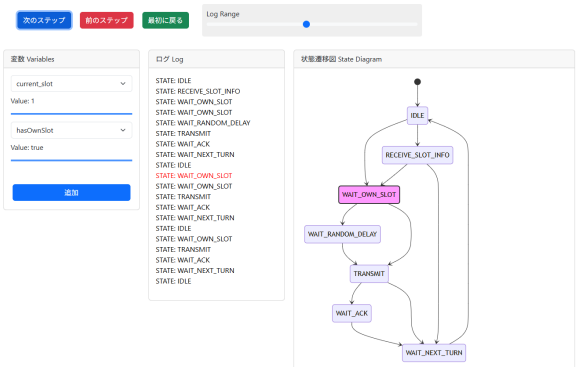

図 8　提案システムによるログ可視化・ステップ実行環境

Fig. 8　Log visualization and step execution environment of the proposed system.

4. 考　　察

本研究で開発したログ可視化システムにより，通信プロトコルの動作過程を逐次的に追跡できることを確認した。これにより，従来のコンソール出力のみに依存したデバッグ手法に比べて，通信動作を直感的かつ体系的に理解することが可能となった。特に，状態遷移図上で現在の状態や変数の変化を視覚的に把握できる点は，デバッグ効率の向上に大きく寄与する。

従来手法では，高速で進行する通信処理の挙動を逐一確認することが困難であり，異常動作の再現や原因特定に多くの時間を要していた。本研究の可視化環境では，ステップ実行機能により状態遷移を一段階ずつ再現できるため，処理の流れや条件分岐を容易に追跡できる。これにより，開発者が異常箇所を迅速に特定できるだけでなく，通信アルゴリズムの学習や設計意図の共有にも有効である。

また，開発した環境は教育的利用や共同研究にも有用であると考えられる。例えば，状態遷移図を教材として利用する際に，図上で通信動作を再現できることは，通信プロトコルの理解促進に寄与する。さらに，複数の開発者がログを共有しながら解析を行うことで，チーム開発時のコミュニケーション効率の向上も期待できる。

一方で，現時点のシステムにはいくつかの課題が残る。第一に，リアルタイム性の向上が挙げられる。現状では，ログを取得後に可視化する「事後解析型」が中心であり，実機動作中の挙動をリアルタイムで完全に再現するには至っていない。第二に，通信性能やデバッグ効率の改善効果を定量的に評価していない点である。定量的な指標に基づく評価を行うことで，提案システムの有効性をより客観的に示すことができると考えられる。第三に，ノード数の増加に伴うログ量の増大や可視化負荷への対応も今後の検討課題である。

総じて，本研究の成果は，通信プロトコルの開発・理解・教育を支援する基盤として有効であることを示唆しており，さらなる拡張によって幅広い応用が期待される。

5. まとめおよび今後の展望

本研究では，状態遷移図から自動生成したファームウェアを汎用ハードウェア上で実行し，無線通信プロトコルの動作を実機で検証可能とする環境を開発した。さらに，通信処理中に出力されるログを解析し，ブラウザ上で状態遷移および変数変化を可視化・ステップ実行できるデバッグ支援システムを実装した。

これにより，従来のコンソール出力によるデバッグでは困難であった逐次的な動作確認を可能とし，プロトコル動作の理解促進と開発効率の向上に寄与することを示した。また，状態遷移図を中心とした設計と可視化環境の統合により，通信プロトコルの設計・実装・評価を一貫して行える開発基盤を実現した。

今後の展望としては，以下の 3 点を挙げる。

（1） **定量的評価の実施：** デバッグ時間やエラー検出効率などを指標として，提案システムの有効性を客観的に評価する。

（2） **大規模ネットワークへの拡張：** ノード数を増加させた環境での通信挙動の検証や可視化負荷の軽減手法の検討を行う。

（3） **リアルタイム可視化機能の強化：** MQTT や Socket 通信を活用したリアルタイムデータストリーム処理により，実時間での挙動確認を可能とする。

また，社会的応用としては，登山者見守りシステムなどへの展開や，通信教育・実習環境への導入が有望である。さらに，将来的には，センサデータの可視化や異常検知アルゴリズムの統合など，応用層への拡張も視野に入れて研究を継続していく。

文　　献

[1] 科学技術振興機構: 「SIP IoE 年次報告書 2022」, 2022.

[2] M. R. Hasan et al.: ”Recent Advancement of Data-Driven Models in Wireless Sensor Networks,” *Technologies*, vol.9, no.4, 2021.

[3] 旭健太、アサノデービッド、不破泰: ”センサーネットワーク検証システムにおける遷移図を用いた MAC プロトコル設計環境の開発”, 信学技報, NS2023-xx, 2022.

[4] 小林遼, アサノデービッド, 不破泰: “センサーネットワーク検証システムにおける遷移図を用いた MAC プロトコル設計環境の開発,”信学技報, NS2023-xx, 2023.

[5]

[6]

[7]

[8]