情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.6　No.1　1–10 (May 2016)

コンシューマ・デバイス論文

IoT機器開発を簡単化するクラウド集約型遠隔制御システムの提案

田添 宏治[1]　南 圭祐[1]　川添 博史[1]　安次富 大介[1]　会津 宏幸[1]

受付日 2015年10月2日, 採録日 2016年2月23日

概要：Internet of Things（IoT）の典型的なアプリケーションの１つは，場所や時間を問わずにセンサや家電等の機器へのアクセスを可能にする遠隔制御である．従来の遠隔制御システムは，機器制御向けに標準化されたローカルネットプロトコルを，専用ゲートウェイを設置することでインターネットに拡張するアプローチと，機器ごとに通信機能を作り込むことで多種多様な機器をクラウドに収容するアプローチの二通りに大別される．前者の課題は専用ゲートウェイの導入・維持コストであり，後者の課題は機器ごとの組込み開発コストである．本論文では，我々は，クラウドとの常時接続に最低限必要な共通機能を搭載した通信アダプタと，通信アダプタが接続するクラウド集約型システムを提案することで両者の課題を同時に解決する．提案する通信アダプタは，シリアル通信をクラウド集約型システムへブリッジングする機能を搭載しており，個別開発を要さずに多種多様な機器をクラウドへ収容する．提案するクラウド集約型システムは，通信アダプタからブリッジングされた機器のシリアル通信を，WebSocket を用いてユーザの操作端末へ中継する機能を搭載しており，専用のゲートウェイを要さずに通信アダプタと操作端末の双方向通信を実現する．IoT 機器開発者は，機器のシリアル通信を処理するアプリケーションを操作端末上に実装することで，遠隔制御を実現可能である．

キーワード：Internet of Things，遠隔制御，HEMS

A Proposal of Cloud-based Centralized Remote Control System to Simplify IoT Device Development

Koji Tazoe[1]　Keisuke Minami[1]　Hiroshi Kawazoe[1]
Daisuke Ajitomi[1]　Hiroyuki Aizu[1]

Received: October 2, 2015, Accepted: February 23, 2016

Abstract: One of typical applications of IoT (Internet of Things) is remote control. The remote control enables users to monitor or control devices (e.g., sensors, home appliances, factory equipment). Previous remote control systems can be classified into two types of architectures. One type of the architectures comprises devices which use a standard local area network protocol and a gateway which extends the local area network protocol into the Internet. This architecture has high introduction and maintenance costs of the gateway. The other type of the architectures comprises devices which can directly communicate with cloud. This architecture has high development cost of communication functions of each of devices. In this paper, We propose new remote control system architecture which resolves the problems of previous two architectures. The proposed architecture comprises IoT adapters. The IoT adapter has a simple bridge function which forwards payload of communication from a device to cloud server. In proposed architecture, one can develop an IoT device by mounting the IoT adapter on a serial port of each device. One can control the IoT device by developing a controller as a smartphone application which processes serial communication from the device.

Keywords: Internet of Things, remote control, HEMS

1. はじめに

近年注目されている Internet of Things（IoT）の典型的なアプリケーションの１つとして，インターネットに接続

[1] （株）東芝研究開発センターネットワークシステムラボラトリー
Network System Laboratory, Toshiba Research & Development Center, Kawasaki, Kanagawa 212–8582, Japan

されたセンサや家電，工場設備等の機器（以下，IoT 機器と呼ぶ）を，場所や時間を問わずに監視したり制御したりする遠隔制御があげられる．

従来の遠隔制御システムは，ECHONET Lite™ [1] や UPnP™ [2] 等，ローカルネット向け標準プロトコルを，専用ゲートウェイでインターネットに拡張して実現するアプローチが一般的である [3], [4]．一方，こうした標準プロトコルに非対応の，その他大多数の機器を遠隔制御可能にする方法として，Arduino™ [5] 等の汎用通信アダプタを用い，機器と通信アダプタ間の通信処理を個別に組込み開発する潮流がある [6], [7]．前者は，標準プロトコルによって，機器側への機器固有の組込み開発を必要としないメリットがあるが，反面，ローカルネットを前提としたプロトコルの性質上，インターネットへの接続を中継する専用ゲートウェイが必要となる．一方，後者は，機器と汎用通信アダプタ間の通信処理を作り込むことによって，標準プロトコルにとらわれない多種多様な機器を，専用ゲートウェイなしで遠隔制御できるメリットがあるが，機器側，すなわち汎用通信アダプタ上への個別組込み開発が必要となる．

我々は，上記 2 つのアプローチが排他的に備えるメリットの両立を課題としてとらえ，遠隔制御を実現する新たな IoT 機器向け通信アダプタ（以降，IoT アダプタと呼ぶ）とクラウド集約型システムを提案する．提案システムを構成する IoT アダプタは，機器との通信をシリアル通信（UART: Universal Asynchronous Receiver Transmitter）に限定し，クラウドとの常時接続を前提とした最小の共通機能のみを搭載する．具体的には，遠隔制御にかかる通信機能として，インターネット側の常時接続通信と UART 通信のブリッジング機能のみを搭載する．UART 上を流れる機器固有のプロトコル処理は，上述した 2 つの既存アプローチとは異なり（**図 1**，**図 2**），スマートフォン等の上で動作する制御アプリケーション（以降コントローラと呼ぶ）側に実装する（**図 3**）．UART は，ECHONET Lite™ 対応機器における機器と通信アダプタ間の標準インタフェースであり，かつ，その他のセンサや機器も，UART を搭載しているものが多い．したがって提案システムは，IoT アダプタ上への個別組込み開発を必要とせずに，多種多様な機器を専用ゲートウェイなしでサポートできるものとなる．

また我々は，UART で制御可能な ECHONET Lite™ 機器（以降，こうした標準通信プロトコルに準拠した機器を標準機器と呼ぶ）と，UART 上の独自プロトコルで制御可能な非標準機器を，単一の IoT アダプタで遠隔制御できることを，パブリッククラウドを用いて実証する．検証システムでは，各機器に特化した通信処理は，コントローラアプリとして実装する構成をとり，Web ブラウザ上で動作する Web アプリとして JavaScript™ で簡単に実現できることを示す．

最後に，提案システムにおける大規模接続時の応答性能をパブリッククラウド上で定量的に評価する．

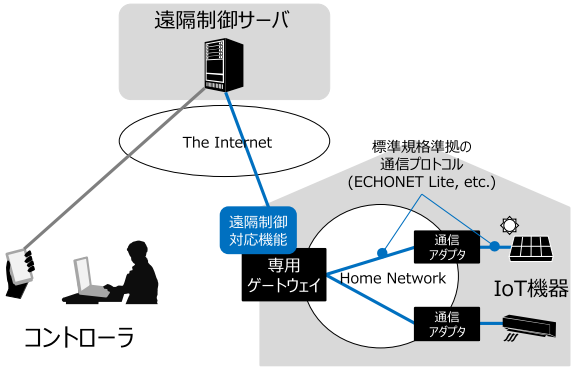

図 1　ゲートウェイ型アーキテクチャ
Fig. 1　Gateway architecture.

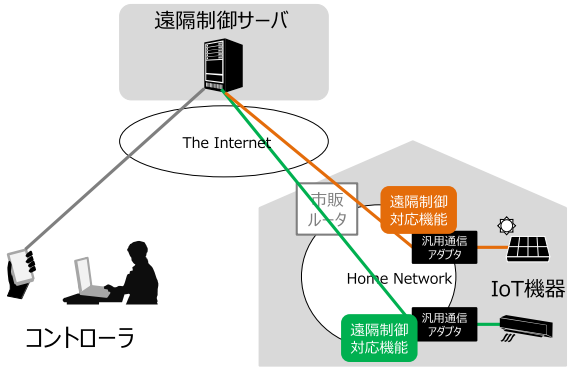

図 2　汎用アダプタ型アーキテクチャ
Fig. 2　Direct connection architecture.

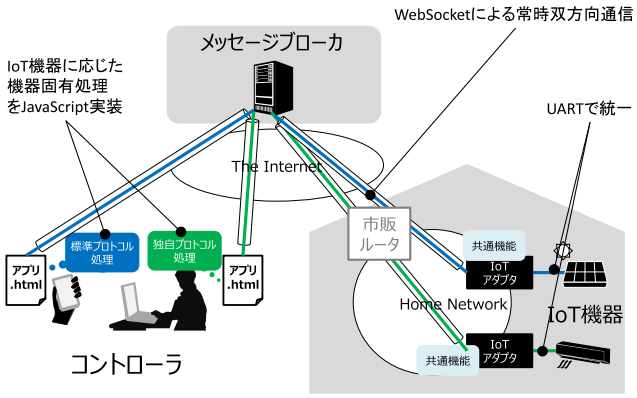

図 3　提案するクラウド集約型システムの概要
Fig. 3　Proposed architecture.

本論文の貢献を要約すると以下のとおりである．

- 機器の通信アダプタへの個別組込み開発を必要とせず，多種多様な機器を専用ゲートウェイなしでサポート可能な，新しい通信アダプタおよびクラウド集約型遠隔制御システムを提案する．
- 実際に提案システムを開発し，標準機器と非標準機器を，単一の通信アダプタを用いてシームレスに制御できることを実証する．
- 提案システムが，大規模接続時にも応答性能の面で実

用上問題ないことをパブリッククラウド上の評価によって定量的に実証する．

以降，2 章で関連研究について述べ，3 章で提案システムの要件と基本設計について記す．4 章で検証システムを試作し，基本的な実現性検証を行い，開発コストの低減効果について論じる．5 章で提案システムの定量的評価を行う．

2. 関連研究

遠隔制御に関する研究は，古くはホームオートメーションをモチーフとしたものを中心に数多く存在する [8], [9]．近年は，IoT の潮流を受け，クラウドコンピューティングやウェブ技術を活用した研究が多い [10], [11]．こうした既存研究のシステムアーキテクチャに着目すると，たとえば文献 [4] は，宅内機器間の通信として UPnP™（Universal Plug and Play）を，宅外の遠隔制御用のチャネル確立方法として SIP（Session Initiation Protocol）プロトコルを利用し，宅内と宅外の境界に専用ゲートウェイを配置する構成をとる．これは遠隔制御システムの一典型であり，ホームネットワーク上のプロトコルに ECHONET™ を用いるもの [3] や，ZigBee™ を用いるもの [12] 等，同等構成をとる提案が多数ある．図 1 に示すこの構成（以降，便宜的にゲートウェイ型と呼ぶ）は，機器の通信アダプタを，ECHONET™ 等の標準プロトコル準拠にすることによって画一化できるメリットを備えるが，一方で，ローカルネットワーク向け機器を遠隔からアクセス可能にするための専用ゲートウェイが必要となるデメリットも持つ．具体的には，エンドユーザの視点では，ゲートウェイを導入する初期コストが必要となり，サービス提供者の視点では，ネットワーク構成が複雑になるため，設置やユーザサポート等にかかるサービス維持コストが増大する．

一方，IoT の文脈で，Arduino™ や Raspberry Pi™ [13] 等の汎用通信アダプタを用い，制御対象機器をクラウドに直接収容する取り組みがある．たとえば，文献 [6] は，クラウド上のサーバとの導通性を Raspberry Pi™ 上に実装したソフトウェアによって維持し，遠隔制御可能なロボットを実現している．同様に文献 [7] は，Arduino™ ベースでスマートメータをクラウドに直収するシステムを提示している．図 2 に，この構成（以降，便宜的に汎用アダプタ型と呼ぶ）を遠隔制御システムに適用したアーキテクチャを示す．この構成は，専用ゲートウェイを必要としないため，前述したエンドユーザの初期コストや，サービス提供者のサービス提供コストを低く抑えられるメリットを備える．特に，文献 [6], [7] のように 1 つのタイプの機器を制御する専用システム向けのアーキテクチャとしては実用的である．しかしながら，多種多様な機器を収容する遠隔制御システムへの適用を想定した場合，機器ごとに汎用通信アダプタへの組込み開発が必要となるデメリットがある．開発コスト低減のために，統合開発環境を提供したり，汎用 OS（Linux 等）を採用したりする取り組みが一般的に行われているが，組込み開発を要する点では変わらない．また，汎用 OS を搭載可能な高スペックのアダプタは，センサレベルの廉価機器には適さない．

本研究は，既存研究のゲートウェイ型と汎用アダプタ型がそれぞれ備えるメリットを両立させる新たな取り組みとして位置づけられるものである．

3. 提案システムの設計

我々は，2 章で述べた既存研究の 2 つのアーキテクチャ（ゲートウェイ型と汎用アダプタ型）が排他的に備えるメリットを両立させるために，新しいクラウド集約型遠隔制御システムを提案する．

3.1 要件

上記の両立は，以下の 2 つの要件として定義できる．

要件 1. 提案システムは，IoT アダプタへの個別組込み開発を要さずに，標準プロトコルに準拠した機器と非準拠の機器の両方をサポート可能であること．この要件により，IoT アダプタをハードウェアおよびファームウェアのレベルで画一化でき，量産による単価低減が期待できる．

要件 2. 提案システムは，専用のゲートウェイを必要としないこと．この要件により，ユーザ側の初期導入コスト，システム提供者側のメンテナンスコスト低減が期待できる．

3.2 アーキテクチャの概要

上記を満たす提案システムの概要を図 3 に示す．提案システムの主要構成要素は，IoT アダプタ，コントローラ，および，クラウド側のメッセージブローカであり，それぞれ以下の役割を担う．

- **IoT アダプタ**：遠隔制御を実現するために必要となる共通機能を搭載する．主に機器側との UART 通信とクラウド側のメッセージブローカとの WebSocket 通信をブリッジする機能を備える．詳細は，3.3 節に示す．
- **コントローラ**：IoT 機器を遠隔制御するアプリケーションである．アプリケーションには，遠隔制御を実現するための機器固有の機能を搭載する．詳細は，3.4 節に示す．
- **メッセージブローカ**：IoT アダプタとコントローラ間の通信を中継するリレーサーバである．通信プロトコルには，TLS [14] ベースの WebSocket [15] を採用した．詳細は，3.5 節に示す．

提案システムは，クラウド上のメッセージブローカの利用を前提とし，遠隔制御機能を共通部分と機器固有部分に分割した上で，それぞれを IoT アダプタとコントローラに搭載する構成をとることで，IoT アダプタの画一化と多様

な機器への対応（要件 1）とゲートウェイレス（要件 2）を達成する．以降，各構成要素の詳細について述べる．

3.3 IoT アダプタ

IoT アダプタの画一化と対応機器の多様性を両立させるために，アダプタ上にさまざまなデバイスに対応する実装を搭載することは，省リソースのハードウェアを前提とする場合には現実的ではない．既存のゲートウェイ型は，機器に標準プロトコル準拠を強制することによって両立を図っているが，我々は，非標準機器の収容も要件に掲げており，この戦略をとることはできない．そこで我々は，提案システムを構成する IoT アダプタを，以下の 2 つの基本方針に則り設計した．

- **アダプタ–機器間の通信媒体の限定**：機器との通信機能を UART に限定する．この設計方針は，対応機器を限定する側面があるが，センサや家電・工場設備等に搭載される組込みシステムの多くが UART 入出力を備えている点と，HEMS（Home Energy Management System）の標準プロトコルの 1 つである ECHONET Lite^{TM} が機器とアダプタ間の通信に UART を採用している点を鑑み，通信媒体の一元化を図っても，標準・非標準を問わず多様な機器をサポートできると判断した．
- **機器固有機能のクラウドサイドへの移譲**：従来アダプタに実装していた機器固有機能をクラウドサイドへ移譲する．この設計方針は，アダプタの主機能を機器側の UART 通信とクラウド側の WebSocket 通信のブリッジング機能に簡略化する．また，機器発見や接続通知等の標準機器・非標準機器を問わず必要となる機能は，クラウドとの連携を前提とした共通機能としてアダプタ上に実装する．図 3 に示すように提案システムでは，アダプタおよびメッセージブローカはコントローラと機器本体の間の通信を中継するだけである．両端点（機器本体とコントローラ）が機器固有のプロトコル処理を生成・解釈し，IoT アダプタやメッセージブローカが通信の中身に関与しない構成を取ることで，提案システムは多様な機器を収容可能になる．

以上に則って設計した IoT アダプタの内部構成は，**図 4** のとおりである．IoT アダプタの機能は，クラウドとの連携により共通機能を実現する通信プロトコル処理（HTTP クライアント）と，コントローラと機器本体の間の通信をブリッジする処理（WebSocket-UART ブリッジ機能）の 2 つによって構成される．この IoT アダプタは，機器と通信アダプタの通信媒体を UART に限定し，機器依存の機能をクラウドサイドに移譲したことによって，自身の画一化，すなわち，ハードウェア・ファームウェアの共通化を達成している．

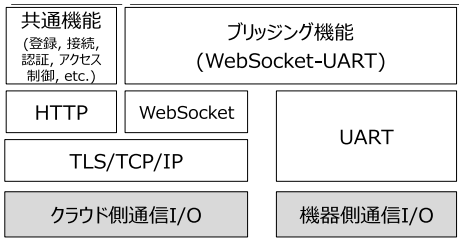

図 4　IoT アダプタの内部構成
Fig. 4　Software stack of IoT adapter.

3.4 コントローラ

コントローラは，IoT アダプタと同様，TLS ベースの WebSocket 接続をメッセージブローカに対して確立し，メッセージブローカおよび IoT アダプタを経由して，機器本体と直接遠隔制御メッセージのやりとりを行う．メッセージの生成と解釈を両端点が担う構成上，機器固有機能は，コントローラ上に実装する．一般的に，コントローラは，スマートフォン・PC 上で動作するネイティブアプリケーションや Web アプリケーションのフロントエンドとして実現される．このため，機器依存機能をコントローラに移譲する本提案システムは，アダプタ上への組込み開発が必要な従来の汎用アダプタ型と比較して，開発や更新等のメンテナンスを容易にするメリットを IoT 機器開発者にもたらす．

3.5 メッセージブローカ

メッセージブローカは，IoT アダプタとコントローラ間の WebSocket 通信を中継する通信サーバである [16]．遠隔制御の対象となる IoT 機器はグローバル IP を持たない環境にあることが多く，クラウド経由の常時双方向通信による遠隔制御を実現するには通信経路上の NAT（Network Address Translation）の存在が問題になる．

我々は，IoT アダプタとメッセージブローカ間の通信に TLS ベースの WebSocket プロトコルを採用することで，この問題を解決する．TLS ベースの WebSocket プロトコルを NAT 透過性確保の手段としたのは，(1) WebSocket のコネクション確立要求は HTTP リクエストであるためファイアウォールやプロキシの影響を受けにくい，(2) 一度確立したコネクションを維持し続けることで通信経路上に NAT が存在する場合でも双方向通信を実現可能である，(3) コネクション確立後はいつでもオーバヘッドの少ない双方向通信が可能である，等の理由による．

メッセージブローカは次のような手順で機器とコントローラ間の通信を中継する．

① 機器（独自プロトコル機器および ECHONET Lite^{TM} レディ機器）のシリアル入出力にそれぞれ IoT アダプタを接続
② IoT アダプタと機器の電源を投入
③ IoT アダプタは，メッセージブローカに自身の持つ固

有 ID 情報を含んだ WebSocket オープニングハンドシェイクを送信し，WebSocket コネクションを確立

④ コントローラは，メッセージブローカに自身の持つ固有 ID 情報を含んだ WebSocket オープニングハンドシェイクを送信し，WebSocket コネクションを確立

⑤ メッセージブローカは，事前に紐付けて登録された ID を持つ IoT アダプタとコントローラの WebSocket コネクションの通信を中継

⑥ コントローラは中継された通信に対して機器固有処理を実行

なお，上記手順の①③は，IoT アダプタに UART 通信と Wi-Fi^{TM} の設定が事前に正しく行われていることを前提としている．また，上記手順の⑤は，IoT アダプタとコントローラの紐付け情報がメッセージブローカに正しく登録されていることを前提としている．これらの前提を実現するために必要なセットアップや紐付け登録等の共通機能は，本論文の主題ではないためここでは詳しく述べない．4 章において構築する検証システムも，これらの機能を搭載しておらず，事前に静的な設定と紐付け登録を行うことを前提としたものである．

4. 検証システムの試作

提案システムの実現性を検証するため，特に，標準機器と非標準機器の双方が単一通信アダプタで透過的に制御できることを検証するため，実際に提案システムを試作した．この検証システムの構成は，**図 5** のとおりである．各コンポーネントの概要は**表 1** のとおりである．本章では，各コンポーネントに関して，3 章との差分となる実装面の詳細について示すとともに，コントローラ上での IoT 対応開発の簡単化について，具体的な対応コードを示し，効果について考察する．

4.1 各コンポーネントの詳細

4.1.1 被制御機器

標準通信プロトコルとして ECHONET Lite^{TM} を用いることとし，標準機器として LED シーリングライトを，非標準機器として UART で AC 100V を制御可能な電源 ON/OFF 装置を用意した．シーリングライトは，本体部と，ECHONET Lite^{TM} 通信機能を付与する ECHONET Lite^{TM} ミドルウェアアダプタで構成されるが，本検証システムでは，このミドルウェアアダプタを IoT アダプタで置き換え，IoT アダプタが本体と接続可能になるようにシリアル入出力を ECHONET Lite^{TM} アダプタ仕様のコネクタに統一した（**図 6**）．一方の電源 ON/OFF 装置は，UART の制御メッセージを信号線の High/Low に変換するデジタル出力ボードと，信号線の High/Low で AC 100V を制御可能なソリッドステートリレーを組み合わせることで作成した．ただし，IoT アダプタが接続可能なよ

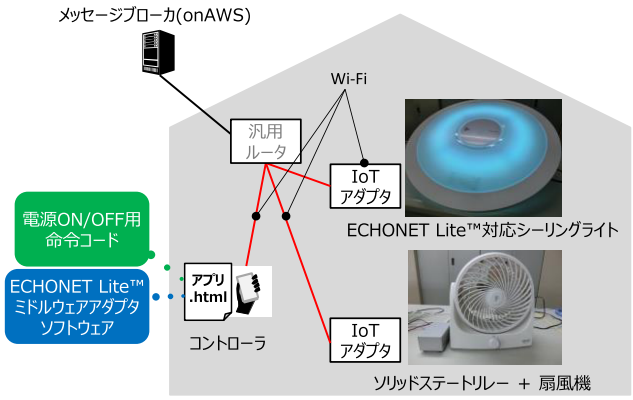

図 5　検証システムの概要
Fig. 5　Demonstration system.

表 1　検証システムの構成要素の仕様一覧
Table 1　Components of demonstration system.

被制御機器		
LED シーリングライト		LEDH82010YXLC-LT1
電源 ON/OFF 装置		
	ソリッドステートリレー	AC100SR
	デジタル出力ボード	AGB65-DD
	レベル変換モジュール	FXMA108
IoT アダプタ		
無線規格		IEEE802.11b/g/n
周波数帯		2.4GHz
無線セキュリティ		WEP，TKIP，AES
CPU		160MHz，32bit RISC，3Mbit SRAM
メッセージブローカ		
インスタンスサイズ		Amazon EC2 m3.medium

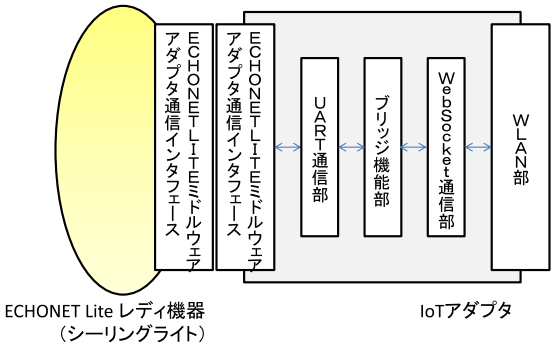

図 6　ECHONET Lite^{TM} 対応機器の内部構成
Fig. 6　Structure of ECHONET Lite^{TM}-compatible device.

うに，デジタル出力ボードの UART 入出力をレベルシフトした上で ECHONET Lite^{TM} アダプタ仕様のシリアル通信コネクタを取り付けた（**図 7**）．検証システムではこの電源 ON/OFF 装置を介して扇風機の電源制御を行う構成とした．

4.1.2 IoT アダプタ

Wi-Fi^{TM} モジュールに組込み向け省リソース WebSocket クライアントを開発し IoT アダプタを試作した．UART は

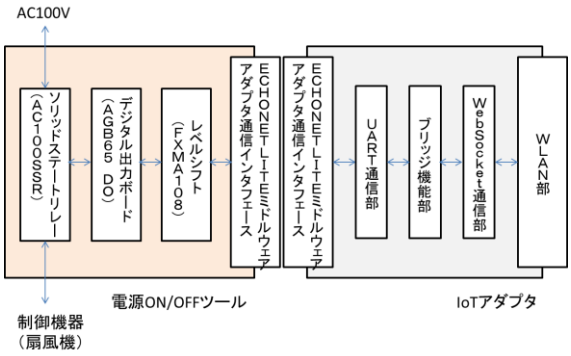

図 7　ECHONET Lite^{TM} 非対応機器の内部構成
Fig. 7　Structure of ECHONET Lite^{TM}-incompatible device.

図 8　Web ベースのコントローラ画面
Fig. 8　Web-based controller.

バイト単位の通信である一方，WebSocket はフレーム単位の通信である．UART から WebSocket への変換において，UART 通信をどこで区切って WebSocket フレームとしてまとめて送信するかを判定する必要がある．検証システムでは 30 ミリ秒の間 UART から新たなバイトを受信しない場合，WebSocket フレームを区切って送信するように実装した．

4.1.3　コントローラ

シーリングライトを制御するための ECHONET Lite^{TM} ミドルウェアアダプタ通信プロトコルと電源 ON/OFF 装置用命令コードを WebSocket の binary フレームを用いて送受信するソフトウェアを，JavaScript^{TM} で実装し，Web ベースのコントローラを試作した（図 8）．サンプルコード 1 およびサンプルコード 2 は，それぞれ，電源 ON/OFF 装置を制御するための JavaScript^{TM} コードとシーリングライトを制御するための JavaScript^{TM} コードのサンプルである．

4.1.4　メッセージブローカ

我々は，メッセージブローカを AWS（Amazon Web Service）上に，メディエータアーキテクチャ（図 9）を採用して構築した [16]．メディエータアーキテクチャは，メッセージブローカを並列化した上で，直接通信をする機器とコントローラが同一のメッセージブローカに収容されるように接続調停を行うことで，スケーラビリティ [16] と低運用コスト [17] を両立することができる．

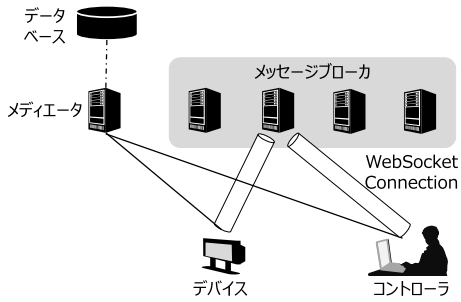

図 9　メディエータアーキテクチャ
Fig. 9　Mediator architecture.

サンプルコード 1　（電源 ON/OFF 装置制御）

```
var createSSR = function (wsURI) {
    var that = {};
    var ws = new WebSocket(wsURI);
    ws.binaryType = 'arraybuffer';
    that.ON = function() {
        var tmpFrame = new Uint8Array( [0xff, 0x8C, 0x01, 0x01]);
        ws.send(tmpFrame.buffer);
    };
    that.OFF = function() {
        var tmpFrame = new Uint8Array( [0xff, 0x8C, 0x01, 0x01]);
        ws.send(tmpFrame.buffer);
    };
    return that;
};
```

4.2　検証結果および考察

検証システムの構築を通じて，3 章で述べた提案システムが，廉価かつ省リソースな IoT アダプタ（表 1）を含めて実現可能であることを実証することができた．定量的な評価は 5 章で述べるが，標準機器・非標準機器を体感的に問題ない遅延時間で透過的に遠隔制御できることが確認できた．以降，本節では，提案システムにおけるコントローラ開発の簡易性について考察する．

まず，電源 ON/OFF 装置のコントローラは，サンプルコード 1 のように WebSocket の binary フレームを用いて命令コードを送信する関数を定義することで実装が可能である（サンプルコード 1）．制御対象である電源 ON/OFF 装置の機能がシンプルな影響もあるが，わずか 15 行程度のサンプルコード 1 を読み込み，createSSR 関数でオブジェクトを生成し，生成したオブジェクトの各メソッドを html のボタンに紐付けるだけで，図 8 のような制御アプリを実現することができる．

次に，シーリングライトのコントローラは，ECHONETLite^{TM} ミドルウェアアダプタソフトウェアをライブラリとして整備することによって，開発を効率化することが可能である（サンプルコード 2）．設計したライブラリの仕様について簡単に説明する．ライブラリでは，createEchonetObject 関数に引数として WebSocket URI を渡して呼び出すことで，ECHONET Lite^{TM} の初期化プロセスを自動で機器に対して実行し，ECHONET Lite^{TM} オブジェクトを生成することができる．この ECHONET Lite^{TM} オブジェクトの set および get メソッドを利用することで機器の遠隔制御が可能である．サンプルコード 2 では，シーリングライトの電源状態を表すプロパティ（0x80）にデータ（[0x30]）を set することで，電源を遠隔で操作する関数と，シーリングライトの RGB 設定に対応するプロパティ（0xf1）にデータを set することで，色を緑に遠隔で変更する関数の例を示した．電源 ON/OFF 装置の場合と同様に．各関数を html のボタンに紐付けると，図 8 のような制御アプリを実現することができる．

提案システムは，次のような理由から従来手法に比べて IoT 機器開発を簡易化することができる．(1) サンプルコード 1 に代表されるように，特定の機器のみでの利用が想定される非標準プロトコルは概して単純であり，簡易なコードでコントローラを実装可能である．(2) サンプルコード 2 に代表されるような，多種多様な機器での利用が想定される標準プロトコルは，複雑ではあるもののライブラリ化することで再利用が可能である．(3) コントローラが Web アプリであることにより新たな機能の追加やアップデートは容易であり，従来手法のように通信アダプタやゲートウェイの組込みファームウェアのアップデートを要さない．

サンプルコード 2　（ECHONET Lite^{TM} 準拠機器制御）

```
var createLED = function (wsURI) {
  var that = {};
  //ECHONET Lite ミドルウェアアダプタを実装した
  //ライブラリが ECHONET Lite ブジェクト生成
  var LED = createEchonetObject(wsURI);
  that.ON = function () {
    //動作状態プロパティ(0x80)に電源 ON([0x30])をセット
    LED.set(0x80,[0x30]).done(function() {
      console.log('set done');
    }).fail(function() {
      console.log('set fail');
    });
  };
  that.GREEN = function () {
    //RGB 設定プロパティ(0xf1)に,
    //RGB(green 値)をセット
    LED.set(0xF1,[0x00,0x00,0x0c,0xb2,0x00,0x00])
    .done(function() {
      console.log('set done');
    }).fail(function() {
      console.log('set fail');
    });
  };
  return that;
};
```

我々は，上記のようにして作成した Web ベースのコントローラ（図 8）の動作確認を行い，体感的に問題のない遅延時間で遠隔制御を実現可能であることを確認した．しかしながら，提案システムは，シリアル通信上のプロトコルをネットワーク経由で運用する構成を取るため，ネットワークの遅延時間が大きすぎると正常に動作しない可能性がある．次章で，我々は，提案システムの応答性能を定量的に評価する．

5. 評価

我々は，検証システムの応答性能を測定し，通信が提案システムを経由する際にかかる時間を評価した．また，大規模接続および負荷が応答性能に与える影響を評価した．

我々は，AWS 上に負荷サーバを追加しホームネットワーク上に IoT アダプタと測定アプリを設置して評価環境を構築した（**図 10**）．この評価環境において，負荷をかけない場合（5.1 節）と負荷をかけた場合（5.2 節）の応答性能を評価した．また，応答性能が遠隔制御の動作に与える影響について述べる（5.3 節）．

5.1 内部処理時間の評価

我々は，検証システムにおいて通信がメッセージブローカを経由する際の遅延時間を測定した（図 10）．ただしここでは，応答性能においてメッセージブローカおよび IoT アダプタが与える影響を評価するため，負荷サーバは利用せずに測定を行った（**表 2**）．

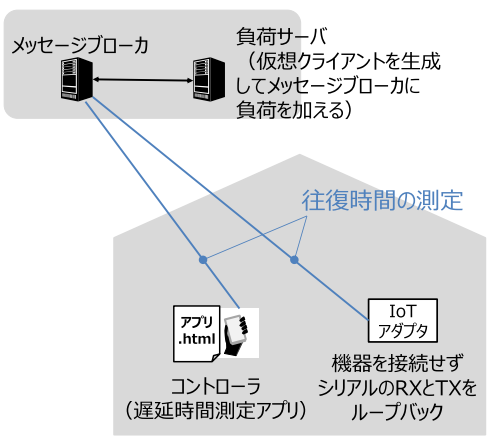

図 10　検証システムを用いた評価環境の構築
Fig. 10　Experimental environment.

表 2　評価環境における遅延時間の測定結果．負荷クライアントの生成した仮想クライアントの数およびすべての仮想クライアントが 1 秒あたりに送信するメッセージの総量を変化させたときの往復時間の平均値・最大値・最小値と 100 ms 以下の往復時間の割合（低遅延率）の変化

Table 2　Experimental results: Relationship between number of virtual client and round-trip time of messaging.

	平均値	最大値	最小値	低遅延率
測定 (1)	53.6 ms	344 ms	46 ms	96.3%
測定 (2)	53.3 ms	110 ms	46 ms	97%
測定 (3)	55.1 ms	280 ms	46 ms	95.1%
測定 (4)	61.9 ms	280 ms	46 ms	93.7%

測定 (1)：制御コマンドが，メッセージブローカを経由してコントローラと IoT アダプタ間を往復するのにかかる時間を測定．ただし，機器の動作時間を測定から省くために，通信アダプタのシリアル入出力を物理的に短絡し信号がただちにループバックするように配線を行ったうえで行った．

一般に，遠隔制御の応答時間は（A. コントローラとメッセージブローカ間の RTT）＋（B.IoT アダプタとメッセージブローカ間の RTT）＋（C. メッセージブローカおよび IoT アダプタ内部の処理時間）＋（D. 機器の動作時間）と見なせる．測定 (1) の値（53.3 ms）は上式の A+B+C に相当する．また，A および B の値は，別途測定したところ，今回の場合は 9 ms であった（検証システムでは IoT アダプタとコントローラは同一のネットワーク環境である）．C のメッセージブローカおよび IoT アダプタ内部の処理時間は約 35.3 ms であることが推測できる．

5.2　負荷による応答性能の変化

我々は，負荷の応答性能に与える影響を次の 3 つの測定で評価した（表 2）うえで，メッセージブローカの接続台数が応答性能へ与える影響と，負荷が応答性能へ与える影響について考察した．

測定 (2)：負荷サーバは 11 万台の仮想クライアントを生成し，すべての仮想クライアントが送信するメッセージの総量が秒間 1,000 メッセージとなるように負荷を加える．

測定 (3)：負荷サーバは 11 万台の仮想クライアントを生成し，すべての仮想クライアントが送信するメッセージの総量が秒間 2,000 メッセージとなるように負荷を加える．

測定 (4)：負荷サーバは 11 万台の仮想クライアントを生成し，すべての仮想クライアントが送信するメッセージの総量が秒間 4,000 メッセージとなるように負荷を加える．

ただし，各仮想クライアントは，1 時間に一度接続維持のためのキープアライブ通信を行うこととした．キープアライブの周期を 1 時間に設定したのは，先行研究 [17] の結果による．また，各仮想クライアントが送信するメッセージは，一律で 512 byte の大きさとした．

文献 [18] は，アプリケーションの応答時間が 100 ms 以下の場合，ユーザはほとんどストレスを感じることなくアプリケーションを利用可能であるとしている．いずれの測定結果においても応答時間が 100 ms を超える割合は少なく，十分低遅延な遠隔制御を実現することができる．

メッセージブローカに接続する仮想クライアントの台数が遅延時間に与える影響について考察する．表 2 において，メッセージブローカへ 11 万台の機器が接続している状態の測定 (2) とメッセージブローカへ機器が接続していない状態の測定 (1) の結果を比較すると，接続台数の多いはずの測定 (2) のほうがいずれの測定結果もわずかながらよい．このことから，メッセージブローカへの接続台数は応答性能に大きな影響を与えないことが分かる．

メッセージングによる負荷が遅延性能に与える影響について考察する．表 2 の測定 (2)，測定 (3)，測定 (4) は同じ数の仮想クライアントをメッセージブローカに接続したうえで，1 秒あたりのメッセージング数を 1,000，2,000，4,000 と変化させたものである．メッセージングによる負荷が増加するにしたがって，100 ms 以上の大きな往復時間が生じる確率が高くなり（したがって低遅延率は低くなり），平均の往復時間も大きくなることが確認された．これは，メッセージブローカの CPU 利用率によるところが大きく，測定 (2) では 1〜3%，測定 (3) では 1〜5%，測定 (4) では 5〜15%であった．CPU 利用率の平均が上がるとともに突発的に負荷が集中する瞬間が増えていくことが観測された．また，大きな往復時間の発生タイミングは CPU の揺らぎに対応していることが観測された．

5.3　遠隔制御の実現性

シリアル通信上のプロトコルには，シーケンスの一部にタイムアウト時間が設定されている場合がある．実際に，本論文で遠隔制御を行った ECHONET Lite^{TM} ミドルウェアアダプタソフトウェアプロトコルにおいても，シーケンスの一部に 300 ms のタイムアウト時間が設定されている．提案システムがプロトコルを正常運用できるかどうかは，このタイムアウト時間をクリアできるかどうかによるところが大きい．測定実験においては，300 ms を超える往復時間が観測されたのは，測定 (1)〜(4) の合計測定回数約 2,600 回のうちわずか 1 回にすぎず，きわめて低確率である．

6.　まとめ

我々は，機器個別の組込み開発や専用のゲートウェイを要さずに多種多様な機器の遠隔制御を実現する新たなクラウド集約型の遠隔制御システムを提案した．また，実際に

提案システムを開発することで，従来，専用のゲートウェイや専用のアダプタを用いて遠隔制御を実現していた機器を，それらを用いずに遠隔制御可能であることを確認した．最後に，開発した試作システムの応答性能を大規模接続を加味したうえで評価し，遠隔制御において事実上問題ないレベルであることをパブリッククラウド上の実験によって確かめた．

参考文献

[1] ECHONET CONSORTIUM, available from ⟨http://www.echonet.gr.jp/english/index.htm⟩

[2] UPnP Forum, available from ⟨http://www.UPnP.org/⟩

[3] Saito, T., Tomoda, I., Takabatake, Y. and Teramoto, K.: Gateway Technologies for Home Network and Their Implementations, *Proc. IEEE International Conference on Distributed Computing Systems Workshop*, pp.175–180 (Apr. 2001).

[4] Kumar, B. and Rahman, M.: Mobility Support for Universal Plug and Play (UPnP) Devices Using Session Initiation Protocol (SIP), *Proc. 3rd IEEE Consumer Communications and Networking Conference* (*CCNC*), Vol.2, pp.788–792 (Jan. 2006).

[5] Arduino, available from ⟨http://www.Arduino.cc/⟩

[6] Prabha, S.S., Antony, A.J.P., Meena, M.J. and Pandian, S.R.: Smart Cloud Robot using Raspberry Pi, *4th International Conference on Recent Trends in Information Technology* (*ICRTIT*), pp.1–5 (Apr. 2014).

[7] Tu, C.-Y., Kuo, W.-C., Teng, W.-H., Wang, Y.-T. and Shiau, S.: A Power-Aware Cloud Architecture with Smart Metering, *Proc. 39th International Conference on Parallel Processing Workshops* (*ICPPW*), pp.497–503 (Sep. 2010).

[8] Liang, N., Fu, L. and Wu, C.: An Integrated, Flexible, and Internet-based Control Architecture for Home Automation System in the Internet Era, *Proc. IEEE International Conference on Robotics and Automation*, Vol.2, pp.1101–1106 (May 2002).

[9] Al-Ali, A.R. and Al-Rousan, M.: Java-based Home Automation System, *IEEE Trans. Consumer Electronics*, Vol.50, No.2, pp.498–504 (2004).

[10] Soliman, M., Abiodun, T., Hamouda, T., Zhou, J. and Lung, C.: Smart Home: Integrating Internet of Things with Web Services and Cloud Computing, *Proc. 5th IEEE International Conference on Cloud Computing Technology and Science* (*CloudCom*), Vol.2, pp.317–320 (Dec. 2013).

[11] Kau, L., Dai, B., Chen, C. and Chen, S.: A Cloud Network-based Power Management Technology for Smart Home Systems, *Proc. IEEE International Conference on Systems, Man, and Cybernetics* (*SMC*), pp.2527–2532 (Oct. 2012).

[12] Kim, K., Park, C., Seo, K., Chung, I. and Lee, J.: ZigBee and The UPnP^{TM} Expansion for Home Network Electrical Appliance Control on the Internet, *Proc. 9th IEEE International Conference on Advanced Communication Technology*, Vol.3, pp.1857–1860 (Feb. 2007).

[13] Raspberry Pi, available from ⟨https://www.raspberrypi.org/⟩

[14] Dierks, T. and Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2, Internet Engineering Task Force, RFC5246 (2008).

[15] Fette, I. and Melnikov, A.: The WebSocket Protocol, Internet Engineering Task Force, RFC6455 (2011).

[16] Kawazoe, H., Ajitomi, D. and Minami, K.: Design and Evaluation of Large-Scale and Real-time Remote Control Architectures for Home Appliances, *IEEE 12th Consumer Communications and Networking Conference* (*CCNC*), pp.820–825 (Jan. 2015).

[17] Ajitomi, D., Kawazoe, H. and Minami, K.: A Cost-Effective Method to Keep Availability of Many Cloud-Connected Devices, IEEE 8th International Conference on Cloud Computing (CLOUD), pp.1–8 (Jun.–Jul. 2015).

[18] Microsoft Windows Dev Center, Plan for performance, available from ⟨http://msdn.microsoft.com/en-us/library/windows/apps/dn391697.aspx⟩

∗ 本論文に記載の商品，機能等の名称は，それぞれ各社が商標として利用している場合があります．

田添 宏治

2014 年東京工業大学大学院理工学研究科修士課程修了．同年株式会社東芝入社．IoT 機器向け通信システムに関する研究開発に従事．

南 圭祐

2010 年大阪大学修士課程修了．同年株式会社東芝入社．家電向け Web システムの研究開発に従事．

川添 博史

2004 年京都大学大学院情報学研究科修士課程修了．同年株式会社東芝入社．家電伝送システム技術，IoT クラウド通信基盤システムの研究開発に従事．

安次富 大介　（正会員）

2002 年株式会社東芝入社．以来，東芝研究開発センターにて，コンシューマ機器向け通信プロトコル，通信システムに関する研究開発に従事．

会津 宏幸

2001 年北陸先端科学技術大学院大学博士後期課程単位取得退学．同年株式会社東芝入社．ホームネットワークおよび Web 技術の研究・開発と標準化活動に従事．