情報処理学会論文誌 Vol.54 No.7 1849–1860 (July 2013)

IEEE1888通信スタックの組込み向け軽量実装

落合 秀也[1,a)] 井上 博之[2] 寺西 裕一[3] 江崎 浩[4]

受付日 2012年11月5日, 採録日 2013年4月5日

概要：インターネット上のセンサデータ配送にHTTP/XML通信が広く使われようとしているが，これはハードウェアやメモリ容量などに制限の多い組込みシステムにとって，挑戦的な課題となっている．我々は，センサデータやアクチュエータ制御コマンドの交換にHTTP/XMLを使用するIEEE1888通信プロトコルに着目し，この通信スタックの軽量実装手法を研究した．本研究で開発したIEEE1888通信スタックは，WRITEクライアント機能が1,820バイト，FETCHクライアント機能が3,360バイトの大きさを持ち，Java，.NET Framework，PHP5，Rubyなどの SOAPライブラリ上に実装されたIEEE1888通信スタックと相互に通信できることが確認されている．

キーワード：機械間通信，XML，IEEE1888

Lightweight IEEE1888 Protocol Stack for Embedded Systems

Hideya Ochiai[1,a)] Hiroyuki Inoue[2] Yuuichi Teranishi[3] Hiroshi Esaki[4]

Received: November 5, 2012, Accepted: April 5, 2013

Abstract: HTTP and XML are getting widely used for sensor data delivery over the Internet. This raises a challenge to embedded systems for limitted memory resources and other hardware reasons. In this paper, we focus on IEEE1888 communication protocol that uses HTTP and XML for exchanging sensor data and actuator-control signals, and show how we can implement it in a lightweight manner. The IEEE1888 communication stack that we have developed in this research has 1,820 bytes for WRITE-client functions and 3,360 bytes for FETCH-client functions. It can communicate with IEEE1888 communication stacks implemented on standard SOAP libraries including Java, .NET framework, PHP5, Ruby.

Keywords: Machine-to-Machine communication, XML, IEEE1888

1. はじめに

2000年代に入ってから，インターネット技術は，センサ機器やアクチュエータ機器との通信媒体としても利用されるようになってきた．この動きは，近年Internet of Things（IoT）[4], [7], [20]やMachine-to-Machine（M2M）[15]と呼ばれる分野に成長し，広域監視制御のアプリケーション（e.g.，施設管理[26]やスマートグリッド[6]）のための基盤技術となりつつある．一方で，Web技術やデータベース技術との親和性を高めるために[11]，インターネット側の通信にHTTPやXMLのような高次な通信プロトコルやデータフォーマットを用いる傾向にある[13], [16]．汎用のHTTPやXML処理ライブラリは多くのフットプリント（動作するのに必要なプログラムやデータがメモリ上に占めるバイト数）を要求するため，組込み機器によっては，実装上の制約となる場合もある．

本研究では，HTTP/XMLを用いる通信方式としてIEEE1888[12], [17], [27]に着目し，この通信プロトコルスタックを，8ビットのワンチップマイクロプロセッサに実

1 東京大学/NICT
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan/NICT, Koganei, Tokyo 184–0015, Japan
2 広島市立大学/NICT
Hiroshima City University, Hiroshima 731–3194, Japan/NICT, Koganei, Tokyo 184–0015, Japan
3 NICT/大阪大学
NICT, Koganei, Tokyo 184–0015, Japan/Osaka University, Suita, Osaka 565–0871, Japan
4 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
a) jo2lxq@hongo.wide.ad.jp

装する技術を研究している．IEEE1888 は，インターネットを介した，センサからの値収集やアクチュエータ制御などを SOAP 通信 [18] で交換するプロトコルであり，データベースや Web などへの連携（i.e.，システム・インテグレーション）をスムーズに行うことを可能にしている．一般に，SOAP 通信を実現するには，Java の Axis2 ライブラリや，.NET Framework のような，フットプリントが，数百メガバイトを持つようなライブラリを利用する必要がある．しかし，実際には，IEEE1888 のように通信規格が定まってしまえば，データ構造が定まるため，必ずしもそのような SOAP 通信ライブラリを使う必要はなく，もっと軽量な実装が可能である．我々は，この原理に基づいて組込みコンピュータ向けに独自の IEEE1888 通信スタックを開発することになった．

組込みコンピュータにも様々な種類があり，たとえば，Linux 系のオペレーティングシステム（OS）を動かすことが可能なもの（e.g.，ARM CPU が使われている）から，OS がほとんど存在していないようなもの（e.g.，ATmega328P で動く Arduino [2] など）まで多様性に富んでいる．我々の研究では，下記の理由で，後者のコンピュータ・プラットフォームをターゲットとし，その環境で動作可能な通信プロトコルの実装手法を研究することになった．

まず，後者のコンピュータ・プラットフォームは，ハードウェアが，部品点数で 50 点，ネットリスト[*1]で 250 行，ピン数が 300 点程度ぐらいで実現可能である．つまり，時間を費やすことなく，基板のパターン設計を行うことができる．CAD を使えば 1 日〜1 週間程度で基板の設計が完了し，さらに 1 週間あれば基板製造や部品実装まで行うことができる．前者のコンピュータ・プラットフォームの場合は，部品点数，ネットリスト行数，ピン数が何倍もあり，開発のハードルは急激に高くなる．

また，後者のコンピュータ・プラットフォームは，ハードウェア・アプリケーション機能（たとえば，温度計測，パルス検出，LED 表示版のダイナミック点灯表示，Pulse-Width-Modulation（PWM）制御など）を直接プログラミングすることができる．Linux 系の OS が動く組込みコンピュータ（たとえば，Armadillo [3]）の場合には，必ずしも周辺 I/O が充実しているわけではないため，専用のアプリケーション・ハードウェアを別途用意し，その間との通信をシリアルで行うなどする必要がある．

本研究は，このような理由から，IEEE1888 通信スタックを，OS がほとんど存在しないようなコンピュータ・プラットフォームに載せることに意味があると判断し，その実装手法を研究したものである．

以下，本論文の構成を示す．2 章で関連研究を取り上げ，3 章で IEEE1888 通信プロトコルのうち，本研究で関係する部分を述べる．4 章で本研究で行った軽量化の手法を整理し，5 章で，実装と評価結果を述べる．6 章で考察およびまとめを行う．

[*1] ネットリストとは回路網の接続関係をリストで表現したものである．小さいほど，回路構成が単純になりハードウェア開発に柔軟性が増す．

2. 関連研究

施設管理やビルディング・オートメーション，あるいはホーム・オートメーションのシステムや機器の設計・開発は，主に産業界で行われている．有名な通信方式としては，Lonworks [14]，BACnet [5]，ZigBee [22]，Z-wave [21] などがあり，組込み機器に実装されて，各所で使われている．ただし，これらの通信方式では，ごく一部の例外を除き，主にバイナリフォーマットのメッセージをやりとりする．

近年，インターネットを活用した遠隔の監視制御が一般的になり，インターネット側に計測データや設備ステータス情報を取り出し，インターネット側から制御コマンドを発行するようなシステム形態が増えてきた．その際に，データベースや Web などと連携をすることになるため，そのエンジニアが扱いやすい，HTTP や XML による通信が規格化されている．その 1 つとして，IEEE1888 [12], [17] があり，また，oBIX [16]，BACnet Web Services [5] などもある．

小型のマイコンに TCP/IP 通信スタックを実装した例は，Dunkels [9] や PICNIC [24] などがある．ただし，TCP/IP の場合は XML 文書と異なり，通信メッセージはバイナリフォーマット化されている．XML 文書を取り扱う場合には，XML の要素名や，名前空間のプレフィックスなどを意識する必要がある．また，HTTP/XML による通信は，Remote Procedure Call（RPC）が通信の基調となるケースも多く，たとえばクライアント機能の実装に徹底するなど，着眼点も異なってくる．なお，最近は組込み向けの TCP/IP ハードワイヤードチップ（e.g.，W5100 [19]）が登場したため，TCP/IP はソフトウェアで実装しなくてもよくなってきた（本研究で使用したハードウェアも W5100 を利用している）が，XML を使うプロトコルに関しては，いまだ組込み向けのチップは存在していないものと思われる．

XML を組込み機器で扱う際に，同等スキーマでバイナリ変換が可能な Efficient XML Interchange（EXI）を使えばよいとする考え方や試みもある [7], [8]．特に，無線通信の場合には XML を EXI 化することで通信量を落とすことが可能である．また，バイナリ表現を使うことにより，プログラムメモリ上の定型文が圧縮できるなどの可能性もある．ただし，EXI を使う場合にはネットワークの途中で，何らかの機器やソフトウェアが，EXI↔XML 変換を行う必要があり，ネットワークの構造に制限を与えるものになってしまう．また，EXI だけで完結するのであれば，本来の XML の利点であるたとえば可読性を失ってしまうことに

もなる．可読性は Web や DB などの開発に関するエンジニアリングコストを大きく下げることに貢献している．

IEEE1888 通信プロトコルスタックの軽量化に挑戦した活動としては，我々の知る限り，mbed や Android 上への実装 [1], [25] がある．しかし，これらの研究は，我々が開発した FIAPUploadAgent（4 章参照）がベースになっている．IEEE1888 プロトコル教科書 [27] に掲載されている FIAPUploadAgent や FIAPDownloadAgent は，本研究で開発された実装と同一のものである．本教科書は，これらライブラリの使い方を解説するチュートリアルとしての位置付けとなっているのに対し，本論文は，これらのライブラリ本体の設計方法，軽量化の工夫内容，評価検証を論じたものとなっている．

3. IEEE1888 通信プロトコル

3.1 システム・アーキテクチャ

図 1 に，IEEE1888 のシステム・アーキテクチャを示す．このシステムは，ゲートウェイ（GW），ストレージ（Storage），アプリケーション（APP）と分類が可能な「IEEE1888 コンポーネント」で構成されており，これらの間で IEEE1888 通信プロトコルでデータ交換が行われる（運用構成に応じて，相互に行われる場合もあれば，一方方向の場合もある）．IEEE1888 は，後に述べる XML メッセージを，SOAP による Remote Procedure Call（RPC）の一連の流れとしてやりとりする通信方式をとっている．SOAP はソフトウェア・オブジェクトを XML に展開し，インターネットを介してやりとりするための通信規格である．IEEE1888 では，SOAP を採用することで，Web 関連技術やデータベースなどの各種情報システムとのインテグレーション性を高くしている．

ここで，この IEEE1888 システムにおける GW，Storage，APP の役割はそれぞれ次のとおりである．

ゲートウェイ（GW）：GW は，センサやアクチュエータのような入出力インタフェースを IEEE1888 で扱えるようにする．統一的な通信プロトコルでセンサやアクチュエータをインターネット・オンライン化する役割を果たす．本研究では「本研究で対象とするプラットフォームの基板上には複数のセンサやアクチュエータが載っているため，これ自身が，それらのセンサやアクチュエータへの GW である」と考える．

ストレージ（Storage）：Storage は，GW によってオンライン化されたデータを長期間にわたって蓄積する．これにより，GW にデータを蓄積できなくても，昨年や一昨年のデータ（たとえば，電力消費状況）を，後から参照可能になる．

アプリケーション（APP）：APP は，その応用方法によって様々な役割および実装形態がある．Storage から読み出してきて，画面や帳票にデータを出力する APP もあれば，読み出したデータを加工分析処理し，その結果を Storage に書き戻すといった APP もある．

ここで，GW，Storage，APP は，総称して IEEE1888 コンポーネントと呼ばれ，それぞれ役割は違うものの，固有の操作インタフェースを持つわけではない．つまり，APP-Storage 間の通信方式と，GW-Storage 間の通信方式は，後述する WRITE/FETCH/TRAP のみで構成されるようにしている．たとえば，APP が Storage から値をとってくるときに FETCH 手順を使うが，GW が，Storage に保存されたアクチュエータの制御信号をとってくるときにも FETCH 手順を使う．

GW，Storage，APP は便宜上の呼び名であり，インタフェースとしてはお互いに接続可能なように規定されている（実際に接続可能かどうかは，実装されている内容およびそれに付随する機能に依存する）．結果として，GW に対しても SOAP+XML を実装することが求められる（注：これは SOAP+XML 通信機能を有することを意味し，必ずしも SOAP サーバにならなければいけないというわけではない）．逆にいえば，GW が SOAP+XML を実装することができれば，これらの IEEE1888 コンポーネント（Storage や APP など）に自由に接続できるようになるため，情報システムとのインテグレーション性が高まるというわけで

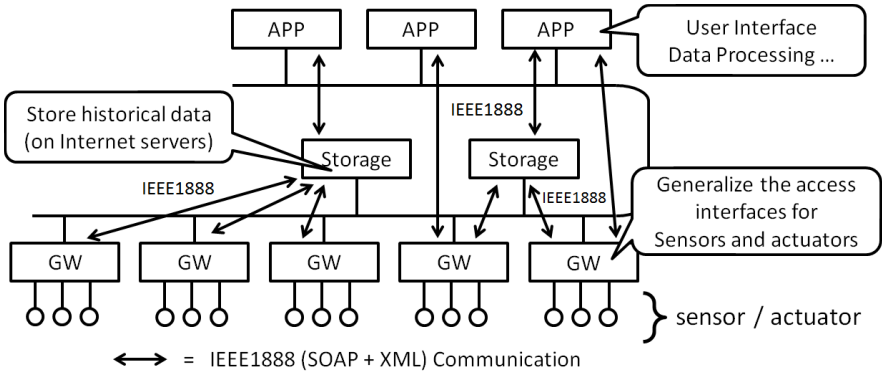

図 1 IEEE1888 のシステム・アーキテクチャ
Fig. 1 IEEE1888 system architecture.

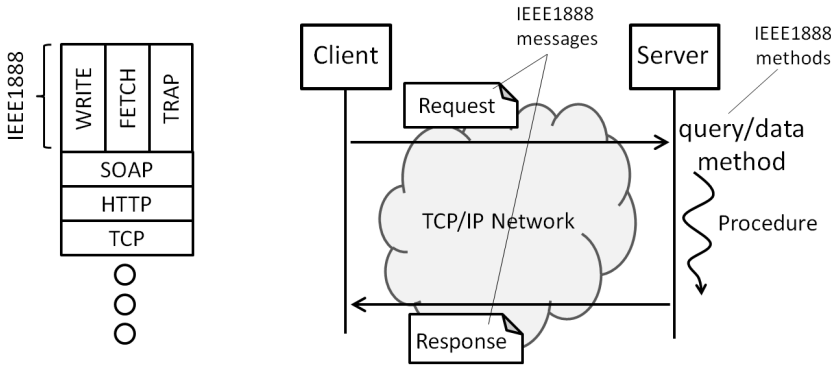

図 2 IEEE1888 通信スタックの構造．(a) IEEE1888 は SOAP 上の通信スタックとして実装される．(b) IEEE1888 は Remote Procedure Call により TCP/IP ネットワークの向こう側にあるサーバの query または data メソッドを呼び出すことで通信を行う

Fig. 2 IEEE1888 protocol stack. (a) IEEE1888 is implemented on SOAP communication stack. (b) IEEE1888 calls query or data methods at server side over TCP/IP network.

ある．

本研究では，GW 実装として，IEEE1888 通信のクライアント機能だけがあれば十分な運用場面を想定し，クライアント機能にフォーカスを絞ることによって，軽量なプロトコルスタック実装を開発するという工夫を行う（4 章参照）．

3.2 通信スタックの構造

図 2 (a) に IEEE1888 通信スタック周辺のプロトコルアーキテクチャ構造を示す．IEEE1888 通信スタックは，HTTP の SOAP 上に実装され，大きく分類して，WRITE，FETCH，TRAP 通信手順が規定されている．IEEE1888 通信は，図 2 (b) に示すように，RPC がベースとなっており，クライアントがサーバに実装された query メソッドや data メソッドを呼ぶことにより，ネットワークを越えた通信（i.e.，WRITE，FETCH，TRAP の実行）が行われる．

IEEE1888 通信手順のうち，本研究で関連するのは，WRITE と FETCH である．WRITE は，ある IEEE1888 コンポーネントが別の IEEE1888 コンポーネントにデータを送り付ける手順（e.g.，GW がセンサデータを Storage に対して送り付ける手順）であり，サーバ側の data メソッドをクライアントが呼び出すことで実行される．FETCH は，ある IEEE1888 コンポーネントが別の IEEE1888 コンポーネントからデータを読み出してくる手順（e.g.，APP が Storage や GW からデータを抜き出してくる，あるいは GW が Storage などから最新の制御コマンドを能動的に取り出してくるときに使う手順）であり，サーバ側の query メソッドをクライアントが複数回呼び出すことで実行される．詳細は文献 [12], [17] を参照のこと．

3.3 実装内容と相互接続に対する考え方

IEEE1888 の GW，Storage，APP は，実際の機器やソフトウェアを開発する段階で，WRITE，FETCH，TRAP のうち，必要な機能だけを実装すればよいことになっている．つまり，画面や帳票に出力するだけの APP であれば，FETCH クライアント機能だけを実装し，Storage から読み出してくることができればよい．また，計測機能しか保有しない機器（GW）で，それが NAT 配下での利用を想定したものであれば，WRITE クライアント機能だけを実装し，Storage に対して定期的に計測データを送信することができればよい．一般に，IEEE1888 においては，各機器やソフトウェアには，実装機能仕様書（この機器は FETCH クライアント機能を持つとか，WRITE クライアント機能を持つなどを定義した文書）が付随し，相互接続は，同手順でのクライアント・サーバ間での接続性を検証することによって行う．つまり，WRITE クライアント機能を持つ GW と，WRITE サーバ機能を持つ Storage との間でデータがやりとりされれば，WRITE に関して相互接続性検証に成功したということができる．

3.4 通信メッセージのデータ構造

IEEE1888 コンポーネント間では，SOAP+XML で要求メッセージと応答メッセージがやりとりされる．SOAP 通信の基礎部分を除くと，IEEE1888 のメッセージは**表 1** に示す 10 種類の XML 要素で構成される．これらを組み合わせて IEEE1888 メッセージは作られる．

以下，本研究に関連する部分の，メッセージの内容を解説する．WRITE の要求・応答メッセージ，および FETCH の要求・応答メッセージをそれぞれ**図 3** と**図 4** に示す．まず，全体のメッセージは，transport で始まり，その transport は，header と body を持てるようになっている．

表 1 IEEE1888 で定義された 10 種類の XML 要素
Table 1 IEEE1888 XML definition (10 XML elements).

XML 要素	意味
transport	メッセージ全体を表す
header	メッセージのヘッダ部分を定義する
OK	処理に成功したことを示す
error	処理に失敗したことを示す
query	読出し処理を対象にすることを示す
key	読出し対象（範囲）を指定する
body	メッセージの内容を定義する
pointSet	ポイントセット（複数のポイントを集約する）
point	ポイントを表現する
value	値（タイムスタンプ情報を含む）

```
<transport xmlns="http://gutp.jp/fiap/2009/11/">
 <body>
  <point id="http://example.org/room010/Temperature">
   <value time="2012-10-20T00:00:00+09:00">25.5</value>
  </point>
  <point id="http://example.org/room010/Humidity">
   <value time="2012-10-20T00:00:00+09:00">58.0</value>
  </point>
  <point id="http://example.org/room010/PowerMeter">
   <value time="2012-10-20T00:00:00+09:00">26992</value>
  </point>
 </body>
</transport>
```

(a) WRITE要求メッセージの例

```
<transport xmlns="http://gutp.jp/fiap/2009/11/">
 <header>
  <OK/>
 </header>
</transport>
```

(b) WRITE応答メッセージの例

図 3 IEEE1888 の (a) WRITE 要求メッセージ，(b) WRITE 応答メッセージの例
Fig. 3 Examples of (a) WRITE request message and (b) WRITE response message of IEEE1888.

WRITE 要求メッセージは，図 3 (a) のような構造を持ち，body 部に，複数の point を並べ，それぞれに value を付与したメッセージである．point の id により，どのセンサのデータであるかを特定することができる．

WRITE 応答メッセージは，header 部に，要求の成否情報が格納されたメッセージである．header 部に，OK が入っていれば成功で，error が入っていれば失敗である．たとえば，図 3 (b) は，WRITE 要求が正しく処理されたことを意味する．

FETCH 要求メッセージは，header 部に query 内容を記載したメッセージである．query 内容には，さらに key として id が埋め込まれ，どのセンサのデータを読み出したいかを指定する．たとえば，図 4 (a) は，温度センサと電力メータの最新値を取得するリクエストである．

FETCH 応答メッセージは，header 部に要求の成否が記載され，body 部に結果が記載されたメッセージである．header 部に query 部が含まれ，ここに cursor が追加されることもある．一方，body 部には，指定したデータが格納されてくる．図 4 (b) がその例である．なお，サーバ側でリクエストの処理中にエラー（たとえば，指定したポイント ID が存在しない）が発生した場合には，`<OK/>`の代わりに，`<error type="POINT_NOT_FOUND">...</error>`が入る．

```
<transport xmlns="http://gutp.jp/fiap/2009/11/">
 <header>
  <query id="9eed9de4-1c48-4b08-a41d-dac067fc1c0d"
         type="storage">
   <key id="http://example.org/room010/Temperature"
        attrName="time" select="maximum" />
   <key id="http://example.org/room010/PowerMeter"
        attrName="time" select="maximum" />
  </query>
 </header>
</transport>
```

(a) FETCH要求メッセージの例

```
<transport xmlns="http://gutp.jp/fiap/2009/11/">
 <header>
  <OK/>
  <query id="9eed9de4-1c48-4b08-a41d-dac067fc1c0d"
         type="storage">
   <key id="http://example.org/room010/Temperature"
        attrName="time" select="maximum" />
   <key id="http://example.org/room010/PowerMeter"
        attrName="time" select="maximum" />
  </query>
 </header>
 <body>
  <point id="http://example.org/room010/Temperature">
   <value time="2012-10-20T00:00:00+09:00">25.5</value>
  </point>
  <point id="http://example.org/room010/PowerMeter">
   <value time="2012-10-20T00:00:00+09:00">26992</value>
  </point>
 </body>
</transport>
```

(b) FETCH応答メッセージの例

図 4 IEEE1888 の (a) FETCH 要求メッセージ，(b) FETCH 応答メッセージの例
Fig. 4 Examples of (a) FETCH request message and (b) FETCH response message of IEEE1888.

3.5 通信文のバリエーション

IEEE1888 は，HTTP と SOAP の通信ルールに従うため，TCP 上を流れるメッセージは，図 3 や図 4 とまったく同一になるとは限らない．たとえば，HTTP には，Transfer-Encoding: chunked 方式で HTTP ボディ部を送信する方式がある．この場合には TCP 上のデータとしては，1 つの XML 文書は途中で分割された形で転送される．また，SOAP では，`<point id="..."` を送る場合に，`<ns:point id="..."` のように XML 名前空間プレフィックスを付与して送信される場合がある．いずれの方式も XML としては正しいのだが，TCP 上のデータとしては異なる形のメッセージとなる．つまり，IEEE1888 の通信スタックとしては，このような違いにも対応できる必要がある．これは概して通信相手の実装に依存してくるものであり，他スタック実装との相互接続性を検証することで，こちら側の通信スタックをテストすることができる（5.3 節

参照）．

4. IEEE1888 通信スタックの軽量化

4.1 目標

本研究でターゲットとするのは，AVR 社のマイクロプロセッサ（ATmega328P）に WIZNet 社の TCP/IP コントローラ（W5100）を組み合わせて作られる程度の規模の通信装置（Arduino Ethernet として市場に出回っているものを想定）である．16 MHz の CPU クロックで動作し，プログラムメモリ領域が 32 kbyte，データメモリ領域が 2 kbyte，不揮発性メモリ領域に 512 byte を利用することができる．ここには，実際には，他のネットワーク処理（DHCP，DNS，NTP）に関するソフトウェア，設定用のコマンドラインインタフェース（CLI）に関するソフトウェア，不揮発性メモリ（EEPROM）の管理ソフトウェア，各種 I/O 処理（アナログ計測，パルス検出，ダイナミック制御，PWM 制御）に関するソフトウェア，など様々なソフトウェアを実装しなければならない．つまり，IEEE1888 通信スタックとして実装できる部分は，かなり限定されてくる．

そこで，本研究では，具体的には，上述のプラットフォームを想定し，

(1) 本研究で開発する IEEE1888 通信スタックのほかにも，実用面において，ほかに重要な機能が実装できること
(2) 本研究で開発する IEEE1888 通信スタックは，ほかの IEEE1888 通信装置と問題なく相互接続通信できること

を達成することを目標とする．

4.2 実装機能の選択

本研究では，以下に述べる機能にフォーカスを絞って，IEEE1888 通信スタック軽量化の工夫を行った．

要件 1：センサ計測データ（ボード上にある数個のセンサのデータ）を，インターネット上の IEEE1888 サーバに送信可能であること

要件 2：インターネット上の IEEE1888 サーバから，アクチュエータ制御に反映させるデータを取得可能であること

IEEE1888 規格の言葉でいえば，**要件 1** は，WRITE クライアント機能の実装であり，**要件 2** は FETCH クライアント機能の実装である．ここで，それぞれの機能は，サーバへのデータのアップロード機能，サーバからのデータのダウンロード機能と考えることができることから，これらのソフトウェアを **FIAPUploadAgent** および **FIAPDownloadAgent** と呼ぶことにする．なお，FIAP（Facility Information Access Protocol）は，IEEE1888 の別名で，ここではコードネームとして使用している．

IEEE1888 通信スタックは，本来 SOAP 上に実装され，HTTP や SOAP の通信ライブラリは別途用意されるものであるが，本研究では，FIAPUploadAgent と FIAPDownloadAgent の中に，必要最小限の HTTP や SOAP に関する処理も一緒に実装することにする．

なお，FETCH では，指定した IEEE1888 ポイント 1 個の最新データのみを取得するものとし，IEEE1888 で規定されている cursor 機能の実装はしなくて済むようにする．つまり，FETCH 手順は 3.2 節で複数回の query メソッド呼び出しであると述べたが，この条件下では 1 回の呼び出しで完結する．また WRITE 手順も 1 回の呼び出しで済む．これにより，実装する処理量を減らすことができる．

4.3 ライブラリの API 設計

前節でフォーカスを絞った機能を実装するために必要十分なアプリケーション・プログラミング・インタフェース（API）を次のように設計した．なお，ここで定義されている API は文献 [27] で紹介されているものと同一である．本論文では，その API 設計について解説するとともに，エラーの扱い方についての工夫を述べる．

4.3.1 FIAPUploadAgent

表 2 および**表 3** に，FIAPUploadAgent が持つ API を示す（これは Arduino の環境を想定した表現方法である）．IEEE1888 サーバの場所を指定して begin メソッドを呼び出した後，post メソッドで送りたいデータを送信する．データは，fiap_element 構造体の上で日時情報とともに定義される形式となっている．

4.3.2 FIAPDownloadAgent

表 4 に，FIAPDownloadAgent が持つ API を示す．IEEE1888 サーバの場所を指定して begin メソッドを呼び出した後，get メソッドで取得したいポイント ID と結果格納用のメモリ領域を指定する．IEEE1888 サーバに対しては，指定されたポイント ID の最新データのみを返答するようにリクエストを発行する．成功すると get メソッドの戻り値が，FIAP_DOWNLOAD_OK となり，値とその値の時刻情報を取得することができる．

4.3.3 エラーの取扱い

FIAPUploadAgent や FIAPDownloadAgent では，TCP/IP 通信レベルの失敗，HTTP（SOAP）通信レベルの失敗，IEEE1888 通信レベルの失敗の違いを判断できればよいと考え，CONNFAIL，HTTPERR，FIAPERR の 3 種類を規定している．実際には，様々なエラーが考えられるが，細かいエラー内容が分かったところで，本研究でターゲットとするコンピュータ環境では，そのエラー情報を活用するすべは持たないことも多いためである．このようにエラー粒度を粗く設定することで，TCP/IP 通信の成否，HTTP 通信の成否（SOAP Fault は HTTP エラー 500 として扱われる），IEEE1888 通信の成否だけを確認すればよくなる．

表 2 FIAPUploadAgent（IEEE1888 WRITE クライアント機能）の API
Table 2 API for FIAPUploadAgent (IEEE1888 WRITE client stack).

メソッド	引数	型	意味
begin	server_ip4	const uint8_t*	サーバの IPv4 アドレスを指定
	server_host	const char*	サーバのホスト名を指定（HTTP ヘッダの Host パラメータに利用される）
	server_path	const char*	IEEE1888 サービスを提供している HTTP サーバのパス
	server_port	uint16_t	サーバの TCP ポート番号
	fiap_id_prefix	const char*	ポイント ID の共通部分をプレフィックスとして指定
	戻り値	void	N/A
post	v	struct fiap_element*	サーバに送信するセンサデータ fiap_element 構造体の配列へのポインタ
	esize	uint8_t	fiap_element 構造体の配列の個数
	戻り値	int	サーバとの通信状態を応答 成功：FIAP_UPLOAD_OK TCP 接続失敗：FIAP_UPLOAD_CONNFAIL HTTP エラー：FIAP_UPLOAD_HTTPERR IEEE1888 エラー：FIAP_UPLOAD_FIAPERR

表 3 fiap_element 構造体
Table 3 fiap_element struct data type.

変数	型	意味
cid	const char*	ポイント ID のポストフィックス（センサごとに個別に指定される）
value	char*	このポイントに結び付けられて送信される値（文字列）へのポインタ
year	uint16_t	送信する値の時刻（年の部分）
month	uint8_t	送信する値の時刻（月の部分）1-12
day	uint8_t	送信する値の時刻（日の部分）1-31
hour	uint8_t	送信する値の時刻（時の部分）0-23
minute	uint8_t	送信する値の時刻（分の部分）0-59
second	uint8_t	送信する値の時刻（秒の部分）0-59
timezone	char*	送信する値の時刻（タイムゾーン表記）

4.4 要求メッセージ送信処理の軽量化

XML の処理において，送信操作は比較的単純で，XML のテンプレートを用意しておき，そこに変数となる値を埋め込みながら送信すればよい．ただし，送り出すメッセージ量を事前に計算して，その大きさ（バイト数）の情報を先に送信する必要がある．以下が実装方法の原則である．この方法を使うと，メモリ消費量を小さく抑えることができる．

作業領域は小さく確保：たとえば，アプリケーション・メッセージを 1 度に確保するために char buf[2048]; のような宣言をしてはいけない（これでは 2 kbyte のメモリ領域を消費してしまう）．確保する作業領域は char buf[50]; 程度に抑える．

細切れに作業する：たとえば，`<transport xmlns="http://gutp.jp/fiap/2009/11/">`という 1 文を送り出すときには，まず，`<transport xmlns=`を送り出す．そして，次に，つづくメッセージとして`"http://gutp.jp/fiap/2009/11/">`を送り出す．このように細切れに作業する．

メッセージサイズ（Content-Length）は事前に推定する：HTTP のボディ部（Content）の長さ（Content-Length プロパティ）を計算するときに，メッセージ全部を作成してから計算するのではなく，送り出されるデータ量（fiap_element の内容）から推定するようにする．あるいは HTTP の Transfer-Encoding: chunked の方式を使う．

定型文はプログラム領域から必要な分だけを SRAM 上に読み出して使う：通常の方法でプログラム内に書き込まれた定型文は，起動時に SRAM 領域にコピーされ，それらが利用される．そのため SRAM 領域を消費してしまう．これを防ぐには，定型文をプログラム領域にのみ配置しておき，必要になったときにのみ，SRAM 領域に一時的に展開させる．

4.5 応答メッセージ受信処理の軽量化

3.4 節で述べたように，応答の通信文は，通信相手となっている HTTP や SOAP スタックによって，バリエーションがある．そこで，これらに対応する必要がある．本研究

表 4 FIAPDownloadAgent（IEEE1888 FETCH クライアント機能）の API
Table 4 API for FIAPDownloadAgent (IEEE1888 FETCH client stack).

メソッド	引数	型	意味
begin	server_ip4	const uint8_t*	サーバの IPv4 アドレスを指定
	server_host	const char*	サーバのホスト名を指定（HTTP ヘッダの Host パラメータに利用される）
	server_path	const char*	IEEE1888 サービスを提供している HTTP サーバのパス
	server_port	uint16_t	サーバの TCP ポート番号
	戻り値	void	N/A
get	pid	const uint8_t*	取得したい IEEE1888 のポイント ID
	value	uint8_t*	取得した結果（文字列値）の格納場所
	n	int	文字列（value）の大きさ
	year	int*	取得した結果の時刻（年の部分）
	month	uint8_t*	取得した結果の時刻（月の部分）
	day	uint8_t*	取得した結果の時刻（日の部分）
	hour	uint8_t*	取得した結果の時刻（時の部分）
	minute	uint8_t*	取得した結果の時刻（分の部分）
	second	uint8_t*	取得した結果の時刻（秒の部分）
	timezone	uint8_t*	取得した結果の時刻（タイムゾーンの部分）
	n_tz	int	文字列（timezone）の大きさ
	戻り値	int	サーバとの通信状態を応答 成功：FIAP_DOWNLOAD_OK TCP 接続失敗：FIAP_DOWNLOAD_CONNFAIL HTTP エラー：FIAP_DOWNLOAD_HTTPERR IEEE1888 エラー：FIAP_DOWNLOAD_FIAPERR

では，まず，HTTP の Transfer-Encoding: chunk 方式に関しては，XML 要素名の途中で HTTP ボディ部が分割されることはないという想定で実装している．また，<の後，6 文字目以内で:が出現したときには，その後を XML 要素名として採用するように，処理している（これにより，XML 名前空間のプレフィックスがある場合とない場合とに耐えられる）．また，FIAPDownloadAgent の場合，リクエスト時のクエリの内容から，value タグは 1 つだけしかないはずなので，出現した value タグがあれば，それが求める結果だと判断するようにしている．

構文解析においては，<で開始されたときに，その後ろが value または error であるかを，まず判断する．それ以外の場合は読み飛ばす．error であれば，結果を FIAPERR として採用する．value であれば，value タグの内部を細かく構文解析する．このようにして受信処理の軽量化を行う．

5. 実装と評価

本研究で開発した IEEE1888 通信スタック軽量版（FIAPUploadAgent および FIAPDownloadAgent）に対して，フットプリントの絶対量，他機能とのフットプリントの比較，対向となる様々な通信スタックとの相互接続性を評価・検証した．なお，本研究では，フットプリントのことを，プログラムメモリ領域やデータメモリ領域の占有量のことを指すものとする．

5.1 フットプリント絶対量

FIAPUploadAgent と FIAPDownloadAgent のフットプリントを次のように検証した．まず，プログラム領域に関しては，通信スタックのソースコードに対し，実装に相当する部分をコメントアウトする場合と，コメントアウトしない場合とで，ビルド後のプログラム量の大きさの差分をとった．なお，この際，TCP 通信スタックに相当する部分は，その差分の中には入っていない（つまり，TCP 通信を実装するプログラム領域は FIAPUploadAgent や FIAPDownloadAgent のフットプリントに含まれないように計算している）．次に，スタック領域，スタティック領域に関しては，通信スタックを実際に稼働させながら，データメモリ（SRAM: static random access memory）領域を適所でダンプすることで，分析評価した．その結果は次のとおりである．

- FIAPUploadAgent のフットプリントは，プログラム領域が 1,820 バイト，スタティック領域が 35 バイト，スタック領域が 109 バイトであった．
- FIAPDownloadAgent のフットプリントは，プログラム領域が 3,306 バイト，スタティック領域が 166 バイト，スタック領域が 310 バイトであった．

実際には，この通信スタック部分だけでは動作できないため，TCP 通信スタックや Ethernet など（DHCP 機能や DNS 機能を含む）のスタックも併用することになる（次節

参照）．

参考までに，ARM9 CPU向けのLinuxで動く，IEEE1888のGWソフトウェアおよびAPPソフトウェア（文献[27]の参照コードとして提供されているサンプルをARM9向けにビルドしたもの）は，それぞれ，60,275バイトおよび59,268バイトのプログラム領域を必要とする．

5.2 他機能のフットプリントとの比較

次に，FIAPUploadAgentとFIAPDownloadAgentを，それぞれ，IEEE1888学習キットとピークカットヘルパー（下記）に実装した．学習キットはセンサのみを搭載する機器（GW）なのでFIAPUploadAgent（WRITEクライアント機能）のみを実装し，ピークカットヘルパーはアクチュエータのみを搭載する機器（GW）なのでFIAPDownloadAgent（FETCHクライアント機能）のみを実装する（これらのライブラリをセンサ計測やアクチュエータ制御にどのように利用しているかは後述する）．この環境下で，それぞれの装置における通信スタックのフットプリント（ここではプログラムメモリ領域に限定する）が，他のソフトウェア機能のフットプリントと比較して，どの程度相対差があるかを調査した．これらの装置（下記）は，2012年10月現在，本研究で開発されたFIAPUploadAgentとFIAPDownloadAgentが実装され，製品として発売されている[10]．

IEEE1888学習キット：この装置は，アナログ入力の温度・照度センサ，デジタル入力（スイッチ）の状態をインターネット上にあるIEEE1888サーバに1分に1回の頻度でタイムスタンプ情報とともに送信する．ユーザは，パソコンをUSBでシリアル接続すると，コマンドライン方式（CLI: Command Line Interface）により，インタラクティブに，この装置のIPネットワーク設定，通信先サーバ（URL）設定，IEEE1888ポイントID設定，NTPタイムサーバ設定ができる．実装しているソフトウェア機能は，Ethernet，DHCP，DNS，NTPクライアント，FIAPUploadAgent（IEEE1888のWRITEクライアント），CLIのコマンドライン処理，時刻管理，EEPROM管理，I/O処理機能などである．なお，ライブラリ群は，Arduino-1.0のものを用いている．

ピークカットヘルパー：この装置は，4桁の7セグメントLED（LED表示板）と圧電ブザーを有し，ネットワークからIEEE1888で取得してきた数値を，LED表示板に表示するとともに，その値があらかじめ設定しておいた警報レベルを超えれば，その警報レベルに応じた警報音を発生させる．ここで7セグメントLEDはタイマが発生させるイベントを使いダイナミック点灯表示する．ユーザは，パソコンをUSBでシリアル接続することで，コマンドラインにより，この装置のIPネットワーク設定，通信先サーバのURL設定，IEEE1888ポイントID設定，データ取得頻度設定，警報レベル設定，警報ガード時間設定ができる．実装しているソフトウェア機能は，Ethernet，DNS，FIAPDownloadAgent（IEEE1888のFETCHクライアント），CLIのコマンドライン処理，EEPROM管理，I/O処理機能などである．なお，ライブラリ群は，Arduino-0022のものを用いている．

図5にそれぞれのソフトウェアが消費するプログラム領域の大きさを図示する．この検証は，それぞれの機能を無効にした場合に，その前後でプログラムメモリ消費量がどの程度変動するかを調べることで行った．

FIAPUploadAgentおよびFIAPDownloadAgentのプログラムメモリ量が5.1節の結果と比べると多いように見えるが，これは，5.1節の結果ではTCP通信スタック量が含まれていなかったのに対し，ここではTCP通信スタック部分を含んでいるためである．

この図から分かるように，FIAPUploadAgentやFIAPDownloadAgentのフットプリントが，他のソフトウェアと比較して突出して大きいわけではない．つまり，十分に軽量化されており，メモリ消費量に関して，他のソフトウェアライブラリと同等のメモリ消費感覚で利用可能なことを裏付けている．

5.3 相互接続性の評価

本研究で開発した通信スタックの機能をテストするために，各種IEEE1888通信スタック実装との相互接続性テストを実施した．この相互接続テストには，2012年3月および10月に東大グリーンICTプロジェクト[23]が主催したIEEE1888相互接続試験などを活用させていただいた．相互接続試験には，WRITEサーバに関しては，Java+Axis2, .NET Framework(C#), C言語で組まれた機器やソフトと，FETCHサーバに関しては，Java+Axis2, .NET Framework(C#)，PHP5，Ruby(+SOAP4R)，C言語で組まれた機器やソフトが持ち込まれた．3.3節の説明のとおり，FIAPUploadAgent（WRITEクライアント機能実装）は，WRITEサーバとの接続性を検証し，FIAPDownloadAgent（FETCHクライアント機能実装）は，FETCHサーバとの接続性を検証した．

- FIAPUploadAgentのテストは，上述の各WRITEサーバに対してセンサデータを送信することで行った．結果としては，すべてのサーバとの接続テストに成功であった．つまり，これらのサーバにデータを送信できることを確認し，例外処理ケースでは，例外を検出できることも確認した．
- FIAPDownloadAgentのテストは，上述の各FETCHサーバに対してセンサデータを送信することで行った．結果としては，すべてのサーバとの接続テストに

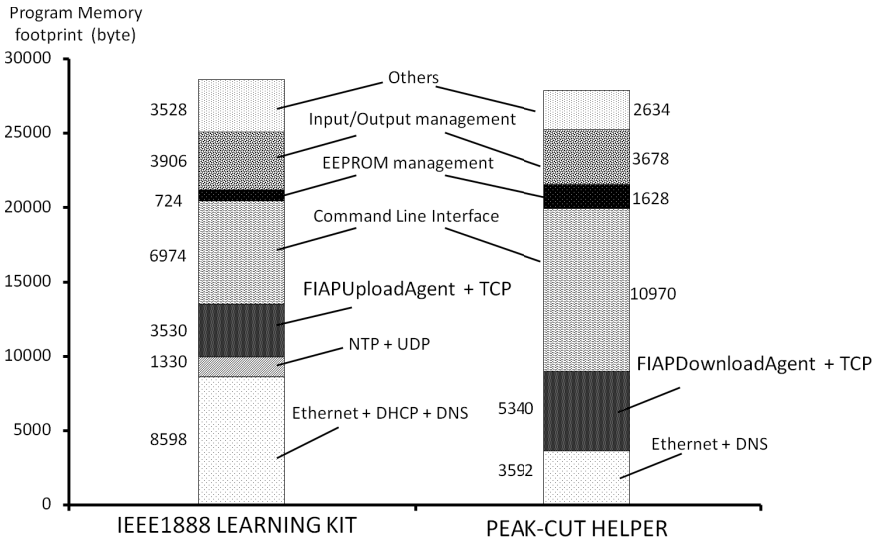

図 5 IEEE1888 通信スタック（FIAPUploadAgent および FIAPDownloadAgent）が占めるプログラム領域の割合．他のソフトウェア機能と比較して突出して大きいことがないことが読み取れる．棒グラフ横の数値は各ソフトウェア機能のフットプリントの大きさ（バイト数）を表している

Fig. 5 Footprint of IEEE1888 communication stack (FIAPUploadAgent and FIAPDownloadAgent). Compared to other software functions, the stack is not remarkably large. The number along the graph shows the footprint (byte) of each software function.

成功であった．つまり，これらのサーバから値を取得できることを確認し，例外処理ケースでは，例外を検出できることも確認した．

ここで，Java+Axis2 の実装は，応答時に，HTTP は Transfer Encoding: chunked モードで応答されていた．また，PHP5 サーバは，応答時に XML 名前空間を XML 要素にプレフィックスとして挿入した XML メッセージを返答するが，そのような状況下でもこれを正しく読み飛ばして，狙いどおりに値をとれることを確認している．

6. 考察

今回の実験では，メモリ使用量の検証をメインに行い CPU 負荷の検証は行っていないが，それは次の理由による．XML 処理では，汎用の XML ライブラリなどを使う場合には，CPU 負荷などを気にしなければならないケースがあるが，本研究で実装した方式では，定型文をプログラムメモリ内に保存しておき，それを細切れにコピーして送信するものであるため，汎用の XML ライブラリと比べて CPU 負荷に関して明らかに軽量である．また，構文解析においても，特定の文字列を意識して抽出する文字列処理であるため，解析処理はほぼ O(m)（m は解析する XML の文字列長）の計算量で達成できる．XML 文字列長はたかだか 1 kByte 程度であり，実際に動かした結果，無視できるほど短時間で処理できていたので，今回のようなケースでは問題にはならないと考えられる．

軽量化という観点では，メッセージ交換に REST/JSON 的な方式や，UDP/IP 方式を用いることも 1 つの方向性としては考えられる．しかし，IEEE1888 がそのような方式を採用していない理由の 1 つとして，JSON では，相互接続性やマルチベンダ性の観点で扱いにくいというものがある．簡易的な情報提供には優れた方式であるのだが，メッセージにオブジェクトのスキーマ情報が載らず，名前空間による各種管理もできないため，上記のような問題が発生する．UDP の上にバイナリメッセージを載せる方式は，設計次第で，さらにコンパクトにできると思われるが，開発効率の問題や，実展開されるネットワークでの問題（Web の一種として扱われないためファイヤウォールが通らない環境が多い）がある．そこで IEEE1888 では SOAP/XML 方式を基調としており，これによって開発効率，展開効率，相互接続性，マルチベンダ性を手に入れたというわけである．SOAP/XML の場合，当然，軽量化に向かないのではないか，という意見も多いと思われるが，今回の研究によって，今回のプラットフォーム（32 kByte プログラムメモリ空間，2 kByte の SRAM 空間）で IEEE1888 機器の実現が可能であることが実証されたわけである．これは，これまでの常識的感覚を見直すきっかけを与えてくれる，と考える．

7. まとめ

我々は，近年，インターネットを介したセンサデータ交換に使われつつある HTTP/XML ベースの通信として，IEEE1888 を取り上げ，この通信プロトコルを，OS がほ

とんど存在しないような組込みコンピュータ（具体的には，Arduino）に実装する手法を研究した．センサデータを送信するために IEEE1888 の WRITE クライアントを，FIAPUploadAgent として実装し，アクチュエータ制御信号を受信するために IEEE1888 の FETCH クライアントを FIAPDownloadAgent として実装した．HTTP や SOAP に関する処理に対応させ，Java+Axis2，.NET Framework(C#)，PHP5，Ruby(+SOAP4R)，C 言語で実装された IEEE1888 サーバと，本研究で開発した IEEE1888 通信スタックが通信できることを確認した．本研究で開発した，FIAPUploadAgent は 1,820 バイトのプログラム領域，35 バイトのスタティック領域，109 バイトのスタック領域を消費し，FIAPDownloadAgent は，3,306 バイトのプログラム領域，166 バイトのスタティック領域，310 バイトのスタック領域を消費する．この大きさは，他の機能実装と比較して突出して大きいことはなく，十分に小型化されているといえる．

参考文献

[1] IEEE1888(FIAP) を mbed で使う，入手先 ⟨http://mbed.org/users/yueee_yt/notebook/fiap/⟩.
[2] Arduino, available from ⟨http://www.arduino.cc/⟩.
[3] Armadillo, available from ⟨http://www.atmark-techno.com/⟩.
[4] Atzori, L., Iera, A. and Morabito, G.: The Internet of Things: A survey, *Computer Networks*, Vol.54, No.15, pp.2787–2805 (online) (2010), available from ⟨http://linkinghub.elsevier.com/retrieve/pii/S1389128610001568⟩.
[5] BACnet – A data communication protocol for building automation and control networks, available from ⟨http://www.bacnet.org/⟩.
[6] Brown, R.: Impact of Smart Grid on distribution system design, *IEEE PES* (2008).
[7] Castellani, A., Bui, N., Casari, P., Rossi, M., Shelby, Z. and Zorzi, M.: Architecture and protocols for the Internet of Things: A case study, *IEEE PerCom* (2010).
[8] Doi, Y., Sato, Y., Ishiyama, M., Ohba, Y. and Teramoto, K.: XML-Less EXI with Code Generation for Integration of Embedded Devices in Web Based Systems, *3rd International Workshop on the Internet of Things* (2012).
[9] Dunkels, A.: Full TCP/IP for 8-Bit Architectures, *ACM MobiSys*, pp.85–98 (2003).
[10] IEEE1888 学習キット，ピークカットヘルパー，入手先 ⟨http://www.futaba-kikaku.jp/⟩.
[11] Guinard, D., Trifa, V., Pham, T. and Liechti, O.: Towards physical mashups in the Web of Things, *IEEE INSS* (2010).
[12] IEEE1888-2011: Standard for Ubiquitous Green Community Control Network (2011).
[13] i.Lon Smart Server, available from ⟨http://www.echelon.co.jp/products/ilon/⟩.
[14] LonMark International, available from ⟨http://www.lonmark.org/⟩.
[15] Niyato, D., Xiao, L. and Wang, P.: Machine-to-Machine Communication for Home Energy Management System in Smart Grid, *IEEE Communication Magazine*, Vol.49, No.4, pp.53–59 (2011).
[16] Open Building Information Exchange, available from ⟨http://www.obix.org/⟩.
[17] Ochiai, H., Ishiyama, M., Momose, T., Fujiwara, N., Ito, K., Inagaki, H., Nakagawa, A. and Esaki, H.: FIAP: Facility Information Access Protocol for Data-Centric Building Automation Systems, *IEEE INFOCOM M2MCN Workshop* (2011).
[18] SOAP version 1.2, W3C Recommendation 2007.
[19] W5100 Datasheet, available from ⟨http://www.wiznet.co.kr/⟩.
[20] Wilde, E.: Putting Things to REST, Technical Report 2007-015, UC Berkeley (2007).
[21] Z-wave appliance, available from ⟨http://www.z-wavealliance.org/⟩.
[22] ZigBee, available from ⟨http://www.zigbee.org/⟩.
[23] 江崎　浩，落合秀也：東大グリーン ICT プロジェクト，電子情報通信学会論文誌，Vol.J94-B, pp.1225–1231 (2011).
[24] 落合正弘：PICNIC の製作，トランジスタ技術，Vol.38, No.1, pp.249–261 (2001).
[25] 大川善邦：mbed+Android データ通信プログラミング，工学社 (2012).
[26] 落合秀也，江崎　浩：グリーン・ビルディングと設備情報アクセスプロトコル：その設計と標準化，通信ソサイエティマガジン，Vol.18, pp.119–126 (2011).
[27] 落合秀也：スマートグリッド対応 IEEE1888 プロトコル教科書，インプレスジャパン (2012).

落合 秀也 （正会員）

昭和 58 年生．平成 18 年東京大学工学部電子情報工学科卒業．平成 20 年同大学大学院情報理工学系研究科修士課程修了．平成 23 年同大学院同研究科博士課程修了．同年同大学大規模集積システム設計教育研究センター助教，現在に至る．博士（情報理工学）（平成 23 年 3 月，東京大学）．設備ネットワーク，広域センサネットワーク，遅延耐性ネットワーク研究のほか，IEEE1888 および ASHRAE での設備ネットワーク標準化活動に従事．

井上 博之 （正会員）

昭和 62 年大阪大学工学部電子工学科卒業．平成 2 年同大学大学院修士課程修了．同年住友電気工業株式会社入社．平成 12 年奈良先端科学技術大学院大学博士後期課程修了．平成 12 年株式会社インターネット総合研究所入社．平成 19 年広島市立大学大学院情報科学研究科講師，現在，同准教授．インターネットアーキテクチャ，コンテンツ配信に関する研究に従事．電子情報通信学会，IEEE 各会員．

寺西 裕一 （正会員）

平成 5 年大阪大学基礎工学部情報工学科卒業．平成 7 年同大学大学院基礎工学研究科博士前期課程修了．同年日本電信電話株式会社入社．平成 17 年大阪大学サイバーメディアセンター講師，平成 19 年同大学大学院情報科学研究科准教授，平成 20 年より情報通信研究機構専攻研究員，招へい専門員を兼任，平成 23 年より情報通信研究機構研究マネージャおよび大阪大学サイバーメディアセンター招へい准教授，現在に至る．博士（工学）（平成 16 年 3 月，大阪大学）．マルチメディア情報システム，ユビキタスサービス，センサーネットワーク，オーバレイネットワーク等の研究開発に従事．本会論文賞を受賞．IEEE 会員．

江崎 浩

昭和 38 年生．昭和 62 年九州大学大学院工学系研究科電子工学専攻修士課程修了．同年（株）東芝入社．平成 2 年米国ニュージャージ州ベルコア社．平成 6 年コロンビア大学客員研究員．平成 10 年東京大学大型計算機センター助教授．平成 13 年同大学大学院情報理工学系研究科助教授．平成 17 年同大学院同研究科教授，現在に至る．工学博士（東京大学）．MPLS-JAPAN 代表，IPv6 普及・高度化推進協議会専務理事，WIDE プロジェクト代表，JPNIC 副理事長，ISOC 理事．