低遅延かつ軽量なセンサネットワーク実現のための技術研究

藤井 彬[†1,a)]　田中 和明[1]

概要：近年 IoT と呼ばれる，センサデバイスなどの組込み機器をインターネットに接続したシステムの活用が注目されている．IoT での通信のために，多くのセンサデバイスが接続されることから，軽量かつ遅延の少ない通信プロトコルが提案されてきている．本研究では，組み込みソフトウェア開発において注目されている mruby を使い， MQTT プロトコルを実装し，従来の http との送信時間と遅延を評価した．

Technology research to realize low-latency and lightweight sensor network

Fujii Akira[†1,a)]　Tanaka Kazuaki[1]

Abstract: In recent years it called the IoT, utilization of the system, which was connected to embedded devices, such as a sensor device to the Internet has been attracting attention. For communication in the IoT, since many of the sensor device is connected, lightweight and low-latency communication protocols have been proposed. In this study, use the mruby that have been noted in embedded software development, implement the MQTT protocol, it was to evaluate the transmission time and the delay of a conventional HTTP.

1. はじめに

近年，モノのインターネット（Internet of Things，以下 IoT）と呼ばれる仕組みが広がりつつある．従来インターネットはパソコンや携帯電話などを使って人間と人間をつなぐことが主な用途であった．しかし，近年ではセンサや無線モジュールの小型化・低価格化・省電力化などの技術的な進化に加えスマートフォンや組込み向け SIM カードなどの通信環境の充実によって，人間ではなく組込みデバイスがインターネットへつながり，モノに取り付けられたセンサが人手を介さずにデータをインターネット経由で利用できるようになった．

現在のところ，IoT での通信には未だに http などの旧来のインターネットで使われてきたプロトコルが利用されている．

しかしながら，組込み機器においては通信の効率化が求められている．今後 IoT デバイスは急激に増加していくことが予想される．多数の IoT デバイスが稼働する上で，ネットワークへの負荷を抑えるために個々の通信量を減らす必要がある．

一方で組込みシステムの高性能化に伴い，製品の付加価値の依存部分はソフトウェアに移行している．組込みシステムにおける開発費用うち，ソフトウェア開発費は全体の 50%に達している．このことから，特にソフトウェア開発の効率を上げることが求められている．

こうした背景から，組込みソフトウェア開発に注目されているのがオブジェクト指向プログラミングである．オブジェクト指向プログラミングは，現在主に使われている C 言語のような手続き型プログラミングと比較して，プログラムコードの凝集度が高まり，結合度が低下することから，開発効率の改善が期待できる．

本研究では，組込みシステム開発で注目されている mruby を用いて，IoT 向けの通信プロトコルである MQTT プロトコルを実装し，自作した mrbgems による MQTT での通信と http での通信に要する応答時間を比較する．

[1] 九州工業大学
[†1] 現在，九州工業大学大学院
[a)] fujii@sein.mse.kyutech.ac.jp

© 2016 Information Processing Society of Japan

2. 本研究で利用する技術

2.1 mruby

mruby は松本行弘氏が開発した Ruby を基により少ないリソースで動作できるようにしたプログラミング言語である．

Ruby は可読性が高くコード量も C 言語で記述した場合の半分で同等の機能を実装できる [2]．そのため，複雑化する組込みソフトウェア開発の生産性効率改善に期待ができる．また，C 言語の API が提供されており，C 言語で記述された既存のリソースを再利用できる．

Ruby との違いは，Ruby は，インタプリタ方式で動作するのに対し mruby は RiteVM という仮想マシン（VM）上でコンパイルした中間コードを実行することで少ないリソースで動作することができる．また，VM を使うことで機器への依存度が低下し，様々な環境へ導入することができる．これにより，作成したアプリケーションを異なるアーキテクチャのマシンで実行したり，実機を使うことなくデバッグ作業を行ったりすることが容易になる．さらにガベージコレクションアルゴリズムにインクリメンタルガベージコレクションを採用することで，Ruby に比べてリアルタイム性が高まっており，組込みシステムに適している．

2.2 MQTT

MQTT(MQ Telemetry Transport) は 1999 年に IBM 社と Eurotech 社により考案されたプロトコルであり，国際標準化団体の OASIS にて標準化が行われている．

MQTT プロトコルは出版-購読型モデルのコネクション指向プロトコルである．このためクライアントは送信者/受信者の区別なく通信をすることができ，1 対多や多対多の通信を行うことができる．

MQTT の通信はクライアントとサーバによって構成される．クライアントはメッセージを送信するパブリッシャ，メッセージを受信するサブスクライバのどちらか，または両方として機能する．サーバはメッセージの中継を行うブローカとしてとして動作する．

MQTT はセンサデバイスなどリソースの限られたデバイスで稼働し，低帯域で信頼性の低いネットワーク上での通信を想定している．

2.2.1 MQTT の通信フロー

MQTT の通信はブローカへの接続確立から始まる．クライアントはブローカへ識別子などの設定を送信し，ブローカがそれに対して返答することで接続が確立する．

次にメッセージを受信するために，サブスクライバはブローカへ Topic を登録する．ブローカはメッセージを受信するとそのメッセージの Topic を登録しているサブスクライバへメッセージを配信する．

メッセージを送信する際には，パブリッシャはデータと Topic，そして QoS(Quality of Service，2.2.2 で詳述) を指定してメッセージをブローカへ送信する．ブローカは QoS に応じて応答を返し，メッセージをサブスクライバへ配信する．

2.2.2 MQTT の特徴

MQTT の特徴を以下に示す．

軽量なヘッダ

固定長ヘッダが 2byte と短く軽量である．

また，http のようにプレーンテキストでの情報ではなく，ビット単位での情報によりオーバヘッドが少ない．

リアルタイム性

コネクション指向のプロトコルであるため，ハンドシェイクなどによるオーバヘッドが少ない．データはブローカからプッシュされてくるので即時性が高い通信が可能．

Topic

要求-応答型モデルであり，Topic を用いることで双方向で 1 対多，多対多の通信が可能．

Topic にワイルドカードを指定でき，柔軟性が高い．

QoS

アプリケーションの特性に合わせてメッセージの到達保証を設定できる．

確実に 1 回，少なくとも 1 回，最高 1 回から選択できる．

Retain

メッセージの配信後に Topic を登録しても受信できる．

Durable subscribe

切断後に再接続しても，切断中のメッセージを受信できる．

Will

クライアントが切断された際に，登録しておいたメッセージをブローカが代理で配信する．

3. 研究内容

3.1 実装対象

本研究では実装対象マイコンとして Raspberry Pi 2 と enzi Board を使用した．

Raspberry Pi 2 は Linux ディストリビューションである Raspbian の上で mruby の環境を構築し，プログラムを実行した．

enzi Board では mruby 実行環境がファームウェアとして搭載されているため，ライブラリをファームウェアに追加した上で，コンパイルした中間コードを micro SD カードに書き込み，実行した．

表 1 に使用したマイコンの性能の比較を示す．

© 2016 Information Processing Society of Japan

表 1　実装対象マイコンの性能
Table 1　Performance of microcomuters

ボード	Raspberry Pi 2	enzi Board
CPU	ARM Cortex-A7 (900MHz)	ARM Cortex-M4 (168MHz)
RAM	1GB	1MB
Middleware	Raspbian	mruby firmware

表 2　MQTT の主なメソッド
Table 2　Methods of MQTT

メソッド名	説明
initialize	ブローカのアドレス，ポートの設定
connect	ブローカへの接続
subscrb	Topic の登録
publish	メッセージの送信
get	メッセージの受信
parse	メッセージの構文解析

表 3　応答時間の比較
Table 3　difference of response time

プロトコル	データ量	平均 [ms]	最大 [ms]	最小 [ms]
http	1 byte	430	430	430
	10 byte	429	430	425
	50 byte	433	435	430
	100 byte	430	430	430
MQTT	1 byte	17.2	17.4	17.0
	10 byte	20.1	20.4	20.0
	50 byte	34.5	42.5	32.5
	100 byte	49.4	55.0	48.5

3.2　ライブラリの実装

MQTT の仕様 [3] に従って mruby のライブラリシステムである mrbgems を作成した．

MQTT は TCP ソケット上で動作するため，TCPSocket クラスを継承した．Raspberry Pi 2 では iij/mruby-socket[4] に含まれる TCPSocket クラスを，enzi Board では組込みライブラリに含まれている TCPSocket クラスを利用した．

作成した mrbgems における MQTT クラスの主なメソッドを**表 2** に示す．

4.　応答時間の評価

MQTT と http で通信の応答時間を比較した．それぞれの通信プロトコルでデータを送信してから受信するまでの時間を計測した．マイコンボードには enzi Board を使い，データの送信から受信までを GPIO のオン・オフに変換し，オシロスコープで時間を計測した．計測の諸元を**表 3** に示す．

図 1 に MQTT と http の応答時間を比較を示す．

http と比較して MQTT での応答時間は 12%以下になった．

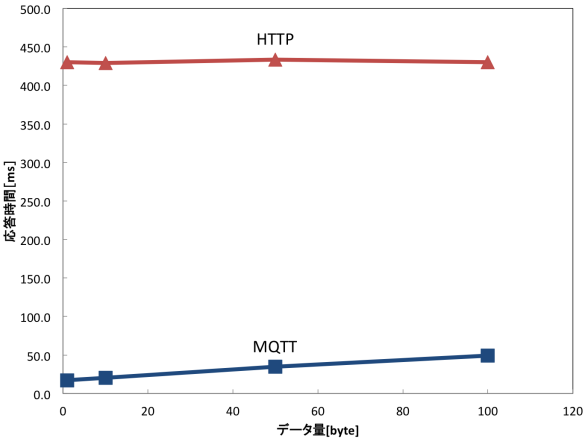

図 1　MQTT と http の応答時間
Fig. 1　Response time of MQTT and http

http と MQTT で大きく応答時間に差が出たのはヘッダによる通信量と通信モデルによる処理時間の 2 つの原因がある．

ヘッダによる通信量

http ではヘッダ部分の情報はそのまま文字列が格納されている．そのため，ヘッダ部分のデータ量大きくなってしまう．

また，http はもともとブラウザ等を用いて様々なコンテンツをやり取りすることを目的としている．そのため，リクエストヘッダにはコンテンツの種類やブラウザの種類について，レスポンスヘッダにはキャッシュの可否などの Web ブラウザが扱うべき内容が含まれている．これらの情報は Web ブラウザでコンテンツのやり取りを行うためには必要であるが，組込み向けの通信においては不要な情報である．結果として http のヘッダのデータ量は 140byte 程度になっている．

MQTT ではヘッダのデータ量は小さくなるように設計されている．メッセージに付加されるヘッダの情報はフラグとしてビット単位で表されている．そのため，MQTT のヘッダは 10byte 程度であり，非常に軽量になっている．

通信モデルによる処理時間

http は要求-応答型モデルのプロトコルである．要求-応答型モデルでは情報のやり取りにはすべて，リクエストとレスポンスが必要となる．つまり，データを送信してから受信するには，それぞれでリクエストとレスポンスが発生するので 2 往復の通信が必要となる．これにより通信回数が増えるため応答時間がかかってしまう．

また，http はコネクションレス型のプロトコルであるため，一回の通信ごとに通信が切断される．http で利用される TCP では，ハンドシェイクによってコネクションの確立と終了を制御する．そのため，1 回の通信には前述し

© 2016 Information Processing Society of Japan

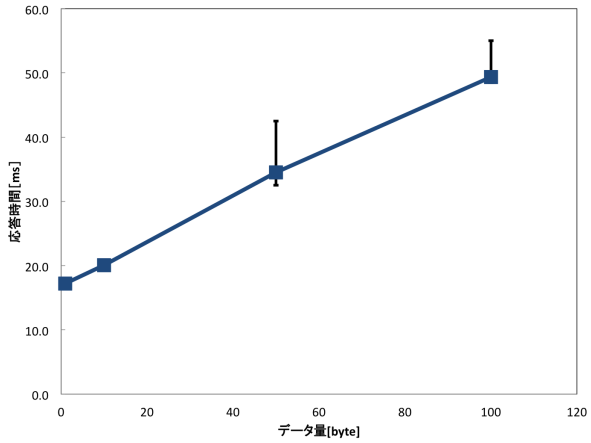

図 2　MQTT の応答時間の変化
Fig. 2　Changes in response time of MQTT

たデータのリクエストとレスポンスに加え開始と終了のハンドシェイクが必要となるため，通信処理にかかるオーバヘッドが更に多くなる．

MQTT ではコネクション指向のプロトコルであるので，TCP のコネクションを維持し続け，データはサーバ側からプッシュされる．よって，データの取得のためにリクエストを送る必要がなくなり，応答性が改善される．また，コネクションが維持されるため，ハンドシェイクによるオーバーヘッドが少なくなる．

図 2 で示されるように MQTT では送信するデータ量に応じて応答時間が長くなっていることから，プロトコル上のオーバヘッドが少ないことがわかる．

5. おわりに

本研究では，組込みシステム開発で注目されている mruby で MQTT プロトコルの基本的な通信を mrbgems として実装した．

その上で，自作した mrbgems と http との通信にかかる応答時間を比較した．MQTT は従来から使われ続けている http よりも通信にかかるオーバヘッドと通信量を抑えることができ，http での通信と比較して 12%の応答時間で通信できることを示した．このことから，MQTT は小さなデータを遅延なく大量に通信するために適しているといえる．

MQTT は http のように汎用的に使うことのできるプロトコルではないが，これからますますの発展が見込まれる IoT 分野において，非常に有用なプロトコルであるといえる．

現在のところ作成した mrbgems はまだ機能が不十分であり，これから仕様に従って更に機能を拡張する必要がある．また，実装方法を改善することで，オーバヘッドを更に小さくすることができると考えられる．

将来的にはこのライブラリをオープンソースとして公開し，IoT 分野への発展に貢献することを目指している．

参考文献

[1] まつもとゆきひろ：まつもとゆきひろ直伝 組込向け Ruby mruby のすべて 総集編，日経 BP 社 (2013).
[2] Sebastian Nanz, Carlo A. Furia: A Comparative Study of Programming Languages in Rosetta Code (2015).
[3] IBM, Eurotech: MQTT V3.1 プロトコル仕様 (2014).
[4] iij/mruby-socket https://github.com/iij/mruby-socket/ (2015).

© 2016 Information Processing Society of Japan