MQTT-SNの実装評価

木村一統[1,a)] 新井イスマイル[2] 藤川和利[2]

概要：IoT デバイスはその特徴によって様々な通信環境下での使用が考えられており、通信環境と使用するネットワーク・プロトコルの組み合わせにより、その通信特性が大きく異なる。このため、通信環境に合わせたネットワーク・プロトコルの選定が重要となる。IoT デバイス向けのアプリケーション層プロトコルに MQTT(MQ Telemetry Transport) と MQTT-SN(MQTT for Sensor Network) がある。本研究では MQTT-SN プロトコルを実装し、MQTT、MQTT-SN の 2 つのプロトコルと、現在最も用いられているプロトコルである HTTP(Hyper Text Transfer Protocol) について、通信環境や QoS の値を変化させた場合に、処理可能なメッセージ数どの程度の影響を受けるのかについて性能比較を行った。結果として、MQTT-SN のアーキテクチャ上の問題点、実装上の問題点が明らかとなった。

Implementation and evaluation of MQTT-SN

Itto Kimura[1,a)] Ismail Arai[2] Kazutoshi Fujikawa[2]

1. はじめに

IoT(Internet of Things) が盛んに唱えられ、多くの注目を集めている。この背景には、小型デバイスの性能比較や省電力化、MVNO(Mobile Virtual Network Operator) の登場により、狭帯域ではあるが、低価格でインターネット通信が行えるようになったという事実がある。IoT ではモノに組み込まれたデバイスにセンサーを取り付け、センシングすることにより、多様なデータを取得し、得られたデータを解析することで有用な情報を取り出そうとする試みが数多く行われている。

現在、IoT 機器が通信を行う場合は、アプリケーション層のプロトコルとして HTTP が用いられることが最も多い。しかし、今後の IoT 機器のさらなる増加に伴うスケーラビリティの問題や、IoT 機器が求められる省電力性を考えた場合、HTTP のヘッダ長の大きさや複雑性から来るオーバヘッドが通信を行う上での問題となる可能性がある。また、IoT に使用されるデバイスは、その組み込み対象のモノが持つ特徴によって、様々な環境下での使用が想定されるため、通信環境と使用するネットワーク・プロトコルの組み合わせにより、その通信特性が大きく異なるものとなる。このため、通信環境に合わせたネットワーク・プロトコルを適切に選定することで、オーバヘッドを軽減し、デバイスのスループットや電力効率を向上させることが可能であると考えられる。

本研究では、IoT デバイス向けの軽量かつ単純なアプリケーション層プロトコルとして注目を集めている MQTT[1] と MQTT-SN[2]、また従来より主要なアプリケーション層プロトコルとして用いられてきた HTTP の 3 種類のプロトコルについて、処理できるメッセージ数を比較する。特に、MQTT と MQTT-SN については、表 1 に示す 3 段階の QoS(Quality of Service) を設定できるため、それぞれの QoS の値を取った時の比較を行った。また、MQTT-SN については今回の実験に必要な機能をすべて備えた実装が存在していなかったため、独自に実装を行った。

実験の結果、MQTT-SN パケットの生成処理内のメッセージ ID の生成およびメッセージ ID とトピック名の管理アルゴリズムでの処理に特に時間を要していることが分かった。

[1] 奈良先端科学技術大学院大学 情報科学研究科 Graduate School of Information Science, Nara Institute of Science and Technology
[2] 奈良先端科学技術大学院 大学総合情報基盤センター Information Initiative Center, Nara Institute of Science and Technology
[a)] kimura.itto.kd3@is.naist.jp

Copyright (c) 2017 by the Information Processing Society of Japan

表 1　MQTT と MQTT-SN における QoS

QoS	通信方式
QoS 0	最高 1 回届ける
QoS 1	最低 1 回届ける
QoS 2	正確に 1 回届ける

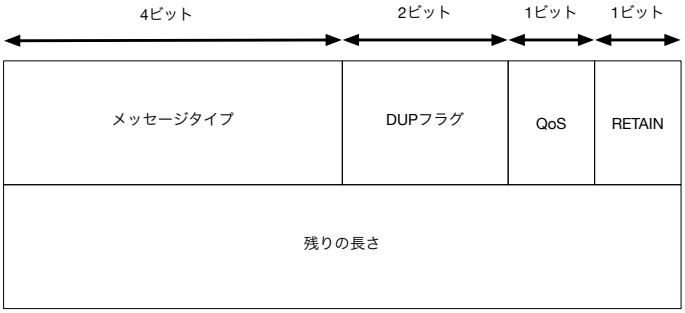

図 1　MQTT ヘッダ

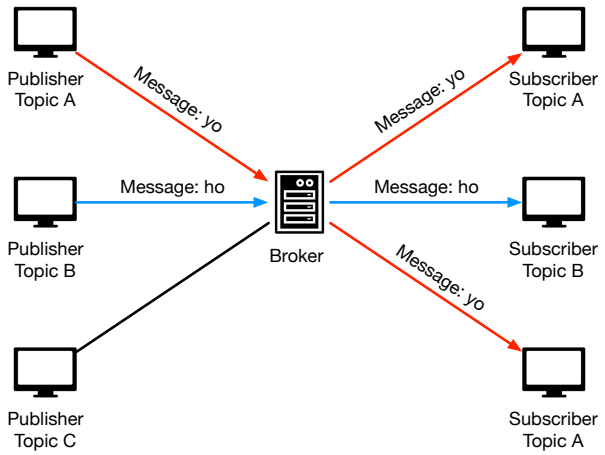

図 2　MQTT アーキテクチャ

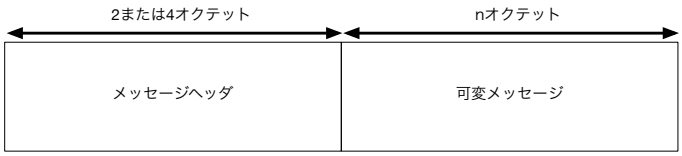

図 3　MQTT-SN ヘッダ

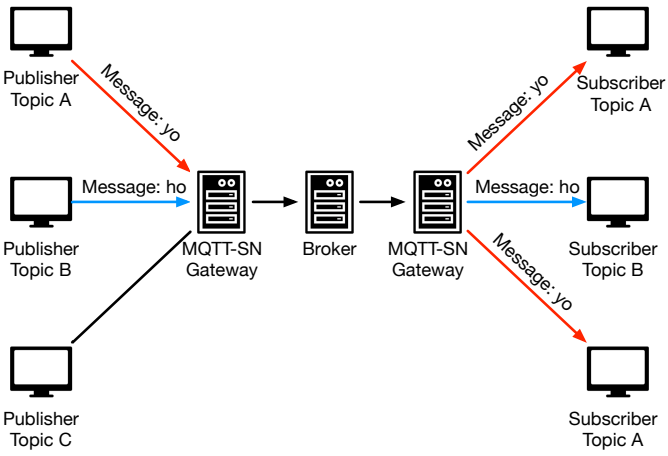

図 4　MQTT-SN ゲートウェイ分離型アーキテクチャ

2. 既存技術

2.1 Pub/Sub アーキテクチャ

MQTT および MQTT-SN はその通信モデルに Pub/Sub アーキテクチャを採用している。Pub/Sub アーキテクチャでは、データの送信者と受信者が直接通信を行うサーバ・クライアントアーキテクチャとは異なり、データの送信者（パブリッシャ・クライアント、以下パブリッシャ）とデータの受信者（サブスクライバ・クライアント、以下サブスクライバ）との間に、ブローカーと呼ばれるサーバを設置し、ブローカーを介して通信を行う。ブローカーの役割はトピック名によるデータ配信先の選定およびデータの転送である。Pub/Sub アーキテクチャでは、サブスクライバは予めブローカーに購読するトピック名を送信し、どのトピック名に対するデータを転送して欲しいのかを指定する。一方、パブリッシャはデータに対してあるトピック名を付与し、そのデータをブローカーに対して送信する。トピック名の付与されたデータを受信したブローカーは、そのトピック名に対し購読予約をしておいたサブスクライバに対してデータを転送する。MQTT、MQTT-SN ではここで述べた単純な Pub/Sub アーキテクチャの備えるべき機能以外にも、ブローカーがメッセージを保持しておく Retain と呼ばれる機能、パブリッシャとブローカー間の通信が途切れた時に送信するメッセージを予め保持しておく Will と呼ばれる機能を持つ。

2.2 MQTT

MQTT は 1999 年に IBM により設計されたアプリケーション層のプロトコル [3] であり、下位層であるトランスポート層のプロトコルには TCP を用いる。プロトコルヘッダは図 1 のようになっている。最小の場合のヘッダ長が 2 オクテットとなり小さいことが特徴である。帯域幅が狭いネットワークやレイテンシの大きいネットワーク、信頼性の低いネットワーク上での使用を目的としている。MQTT では、パブリッシャとサブスクライバといったクライアントとブローカーの間で、MQTT による通信が行われる（図 2）。

MQTT では表 1 に示す QoS が設定できる。これにより、到達保証の度合いが変化する。

2.3 MQTT-SN

MQTT-SN は、MQTT とほぼ同等の機能を有している。大きく異なるのは、下位層であるトランスポート層のプ

Copyright (c) 2017 by the Information Processing Society of Japan

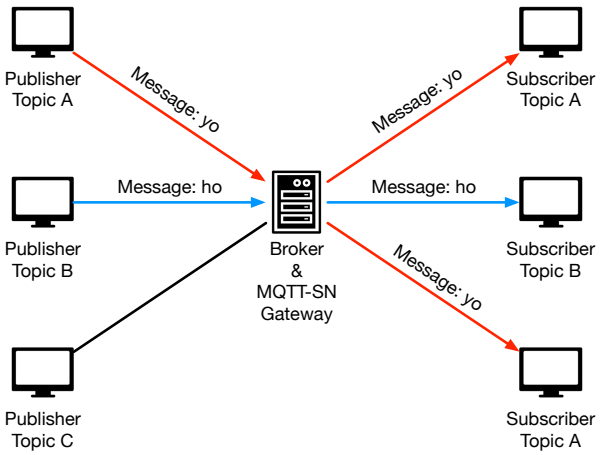

図 5　MQTT-SN ゲートウェイ一体型アーキテクチャ

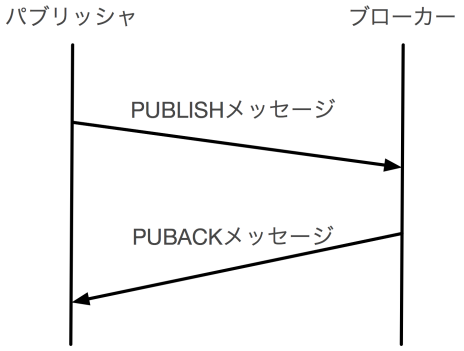

図 6　QoS1 の到達保証

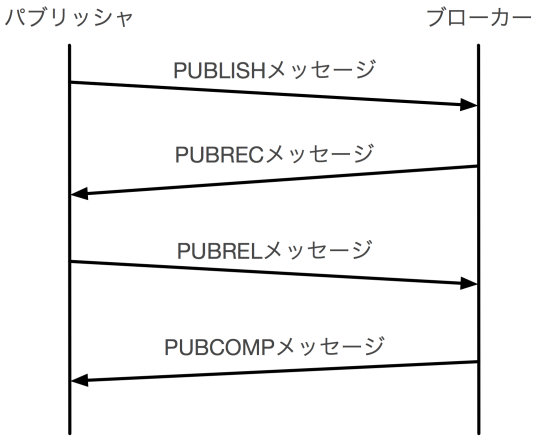

図 7　QoS2 の到達保証

ロトコルに UDP を用いている点である。プロトコルヘッダは図 3 のようになっており、MQTT と同様に最小のヘッダ長が 2 オクテットと小さいことが特徴である。他の MQTT と異なる点としては、MQTT-SN ではセンサー等がスリープ状態に入った場合の動作も仕様上定められており、MQTT よりも更に、省電力なセンサーデバイスへの適用に特化したプロトコルとしている。MQTT-SN では、パブリッシャとサブスクライバといったクライアントとブローカーの他に、ゲートウェイを設置する。ゲートウェイはクライアントから送られてきた MQTT-SN パケットを MQTT パケットに変換し、ブローカーに転送する役割を担う。このため、ブローカーは MQTT で使用したものがそのまま使用できる。この役割を実現するための機能としてゲートウェイには MQTT と MQTT-SN 間でトピック名を変換する機能がある。MQTT ではトピック名として任意の長さの文字列を取ることが可能であるが、MQTT-SN では 2 オクテットの長さまでしか取ることが出きない。このため、ゲートウェイでは MQTT で指定されたトピック名を 2 オクテット内に収まる範囲にまで縮小し、その対応表を保持する必要がある。

また、MQTT-SN によるネットワークのアーキテクチャは、ブローカとゲートウェイを一体化させるか否かにより、ゲートウェイ分離型とゲートウェイ一体型の 2 種類に分けられる。ゲートウェイ分離型（図 4）では、クライアントとゲートウェイ間にて、MQTT-SN による通信を行い、ゲートウェイとブローカー間のネットワークで MQTT による通信が発生する。一方、ゲートウェイ一体型（図 5）では、ゲートウェイとブローカー間の MQTT による通信は、ホストの内部でのみ起こり得る。

MQTT-SN でも MQTT と同様に、表 1 に示す QoS が設定できる。

本研究における実装・実験では、実装の単純化のためにゲートウェイ分離型を採用した。

2.4　到達保証

MQTT と MQTT-SN では、表 1 に示す到達保証を行うために QoS 1 または QoS 2 の場合に確認応答メッセージが発生する。

QoS 1 の時の到達保証の流れを図 6 に示す。始めにパブリッシャから PUBLISH メッセージがブローカーに向けて送信され、PUBLISH メッセージを受信したブローカーがパブリッシャに向けて PUBACK メッセージを返す。パブリッシャは PUBACK メッセージを受信することで PUBLISH メッセージがブローカーへ届いたことを知る。仮に PUBACK メッセージがパブリッシャに届かなかった場合には、パブリッシャが PUBLISH メッセージを再送する。

QoS 2 の時の到達保証の流れを図 7 に示す。始めにパブリッシャから PUBLISH メッセージがブローカーに向けて送信され、PUBLISH メッセージを受信したブローカーがパブリッシャに向けて PUBREC メッセージを返す。この時、ブローカーは PUBLISH メッセージを保持しておくが、「受信した」とは見なさない。パブリッシャは PUBREC メッセージを受信することで PUBLISH メッセージがブローカーへ届いたことを知る。仮に PUBREC メッセージがパブリッシャに届かなかった場合には、パブリッシャが PUBLISH メッセージを再送する。PUBLISH メッセージ

Copyright (c) 2017 by the Information Processing Society of Japan

が再送されてきた場合、ブローカーは保持していた PUBLISH メッセージを捨て、新たな PUBLISH メッセージを保持する。続いて、PUBREC メッセージを受け取ったパブリッシャはブローカーに向けて PUBREL メッセージを送信する。PUBREL メッセージを受信したブローカーはパブリッシャに向けて PUBCOMP メッセージを返し、保持しておいた PUBLISH メッセージを「受信した」と見なす。仮に PUBCOMP メッセージがパブリッシャに届かなかった場合には、パブリッシャが PUBREL メッセージを再送する。このように、ゲートウェイからパブリッシャへの到達通知メッセージ PUBREC についても到達保証をすることで、1 回のみ PUBLISH メッセージを受信することを保証する。

3. 関連研究と課題

MQTT を IoT デバイスへ適用した先行研究として粕谷ら [4] の研究が挙げられる。この研究では建築設備へセンサーデバイスを取り付け運用することを目的として、送信メッセージを 10,000 件/分から 1,000,000 件/分までの間で順次増加させた場合や、QoS の値を変化させた場合のブローカーへの負荷検証を行っている。ブローカーを Cloudn FLAT[5] を上に構築して実験を行っているが、もっとも廉価なプランにおいても平均 150,000 件/分の PUBLISH メッセージが送受信できることを確認しており、実用には十分としている。

Shinho[6] の研究では、有線接続されたネットワークと無線接続されたネットワークにおいて、QoS の値を変化させた場合のパブリッシャとサブスクライバの間のネットワークレイテンシ、パケットの損失率について、それぞれ検証を行っている。この研究では、有線接続と無線接続ともに、QoS の値が上がるにつれ、パブリッシャとサブスクライバの間の遅延は大きくなる結果が出ている。また、パケットの到達数に関しては、QoS の値が上がるにつれ高くなり、QoS 2 の場合がもっとも損失率が低くなるという結果が出ている。

MQTT における通信を行った場合のデバイスの消費電力に関する先行研究として [7] が挙げられる。この研究では Android 端末上で MQTT 通信を行い、その時の消費電力、時間あたりのメッセージ送受信数を 3G 回線および Wi-Fi 回線において計測することを QoS の各値に関して実験している。この結果では、3G 回線よりも Wi-Fi 回線の方が消費電力が小さくなるという結果が出ている。

MQTT を HTTP と比較した先行研究として [8] が挙げられる。MQTT に TLS(Transport Layer Security) を適用し [7] と同様の観点から評価を行っており、それに加え HTTPS による通信との比較も行っている。この研究では、ほとんどの場合において、HTTPS よりも MQTT の方が消費電力が小さく、時間あたりの受信メッセージ数も MQTT の方が多いという結果が出ている。

表 2 端末スペック

CPU	Intel Xeon L5520 2.27GHz
メモリ	8GB
NIC	Intel®82579V Gigabit Network Connection
OS	Ubuntu 16.04

以上の研究成果を見ると分かるように、MQTT については実ネットワークでの検証実験や、他のプロトコルとの組み合わせ実験、消費電力の評価が為されている。また HTTP は現在、IoT デバイスだけでなくネットワーク通信について最も頻繁にしようされるプロトコルであり、多くの研究が為され、実ネットワークでの使用も多くの知見が存在する。

しかし、MQTT-SN については完成度が十分な実装も存在していないため、通信プロトコルに MQTT-SN を採用した例はほとんどない。このため、定量的な評価が行われていない。

2.3 で述べたとおり、MQTT-SN は MQTT と比較して、よりセンサーデバイスへの適用に適したプロトコルとして設計されている。このため、センサーデバイスへの適用を考えた場合に、MQTT 以上に効率的な通信が行えるシチューエーションも考えられる。以上より、通信状況やデバイスの性能によって、MQTT、MQTT-SN、HTTP のうち、どのプロトコルを用いるかを適切に選択することができれば、より効率的な通信が可能となると考えられる。

4. 実装

MQTT に関しては、Eclipse による MQTT クライアントライブラリである Paho[9] ライブラリを用いてプログラミング言語 Python にてパブリッシャとサブスクライバの実装を行った。

MQTT-SN に関しては、プログラミング言語による速度差を軽減するために MQTT と同様に Python にてパブリッシャとサブスクライバ、ゲートウェイの実装を MQTT-SN Version1.1 の仕様に基づいて行った。MQTT-SN アーキテクチャではゲートウェイとブローカー間にて MQTT による通信が発生するが、その部分に関しては Paho ライブラリを用いて実装を行った。また、MQTT と MQTT-SN 間におけるトピック名の対応表については、Python の Dictionary Class にて保持することとした。MQTT-SN のパケットにはメッセージ ID と呼ばれるメッセージを一意に特定する ID を格納する必要があるが、メッセージ ID は乱数として生成し、その管理も Python の Dictionary Class にて行った。

HTTP に関しては、Python の urllib モジュールを用いてクライアントプログラムの実装を行った。

Copyright (c) 2017 by the Information Processing Society of Japan

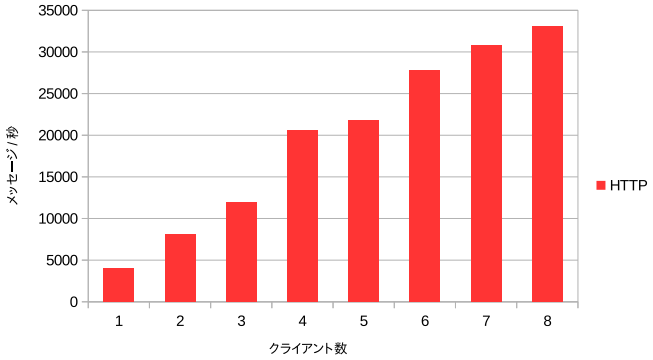

図 8　HTTP の処理パケット数

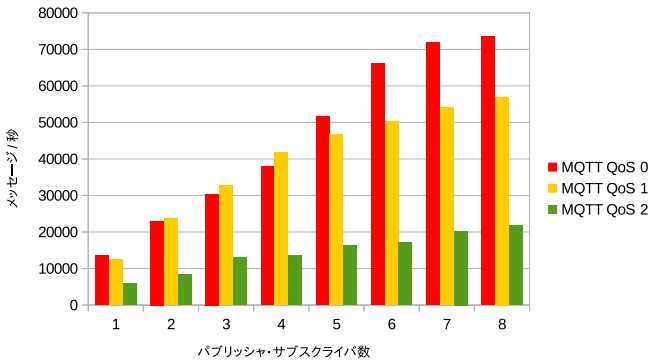

図 9　MQTT の処理パケット数

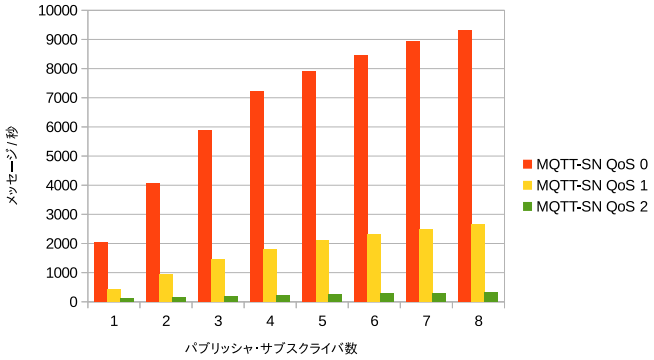

図 10　MQTT-SN の処理パケット数

表 3　関数呼び出しによる経過時間

関数・メソッド名	経過時間 [s]
_generate_msgid	172.968
_recv(multiprocessing/reduction.py)	27.925
__init__(multiprocessing/reduction.py)	25.972
posix.write(built-in)	23.564
packet_formatter	23.564
sendto(_socket.socket)	18.917
_make_packet	18.643
__getitem__(enum.py)	18.394
dumps(multiprocessing/reduction.py)	17.859
publish	15.383

5. 実験

HTTP、MQTT、MQTT-SN の 3 種のプロトコルについて、メッセージに含まれるペイロードの大きさを MQTT-SN Version1.1 の最大ペイロード長である 255Byte として、クライアントまたはパブリッシャ・サブスクライバの数を増やしながら、サーバまたはブローカーで処理できる PUBLISH メッセージの数を計測した。特に MQTT、MQTT-SN に関してはそれぞれ図 2、図 4 のアーキテクチャにおいて計測を行い、Python のプロファイラを用いてどの処理に対して多くの時間が掛かっているのかについても計測を行った。今回の実験では、ブローカーとゲートウェイは同一端末上で動作させ、ゲートウェイの処理を Python の cProfile プロファイラを用いて観測した。MQTT ブローカーとしては Mosquitto[10] を用いた。HTTP に関しては一般的なサーバクライアントアーキテクチャ下での計測を行った。HTTP 通信におけるサーバとしては Apache Version2.4.18[11] を用いた。この実験も予備実験と同様にパブリッシャ、サブスクライバ、ブローカーとゲートウェイとして、表 2 に示すスペックの端末を使用した。

6. 評価

実験の結果、HTTP、MQTT、MQTT-SN においてサーバまたはブローカーが処理できるメッセージ数について、図 8、9、10 に示す結果が得られた。

MQTT の QoS 0 と QoS 1 の場合について、HTTP よりも多くメッセージを処理できていることが分かる。QoS 2 のメッセージ処理数が大幅に低下している原因としては、到達保証パケットが多くなりパブリッシャのパケット送信処理における並列数が上がらないためであると考えられる。

MQTT-SN のメッセージ処理数は HTTP、MQTT と比較して、大幅に下がっている。この理由は大きく分けて 2 つある。

1 つ目の理由は、クライアントとゲートウェイでのパケット生成処理に時間が掛かっていることである。Python のプロファイラによると、数ある処理の中でパケット生成処理に掛かる時間が最も大きかった。表 3 は MQTT-SN の通信において呼び出させる関数・メソッドを、その内部で経過した時間の順に並び替えたものの上位 10 個である。今回は 665 秒間 MQTT-SN クライアント・ゲートウェイを動作させた。関数・メソッド名の後の括弧内はその関数・メソッドが定義されている Python モジュールを表している。括弧がないものは今回ライブラリとして実装した関数・メソッドである。multiprocessing/reduction.py 内で定義される関数・メソッドに関しては、並列化を行う場合に必ず必要なコストであるため議論しない。また、posix.write や sendto に関しても、パケットの送信やプロセス間通信など呼び出し箇所が複数に渡るため今回は議論しない。まず注目すべきなのが_generate_msgid 関数である。この関数で

Copyright (c) 2017 by the Information Processing Society of Japan

はメッセージ ID として用いる乱数を生成し、生成した ID が他のメッセージと重複していないかについてもチェックする。重複した場合については、再度乱数生成を行い、再び重複をチェックする。続いて、__getitem__に着目する。この関数はメッセージ ID の管理や MQTT と MQTT-SN 間でのトピック名の管理において呼び出させる Python の enum モジュールの関数である。その他の関数については以下の通りである。packet_formatter 関数は QoS やメッセージ ID などの値をメモリ上に書き込みパケットを作成するための関数である。_make_packet は各種メッセージにおいて、パケット内に書き込むべき値を計算する関数である。publish はこのライブラリを用いるプログラムが、PUBLISH メッセージを送信する時のフロントエンド関数である。

以上のことから、MQTT-SN において効率的な通信を行うためには次に示す実装上の課題がある。1 つはメッセージ ID の生成方法アルゴリズムに関する課題である。この課題の解決のためには、メッセージ ID 生成に乱数を用いないことや、可能な限り重複しないメッセージ ID を生成できるようなアルゴリズムを用いて、メッセージ ID の重複が起こり得る回数を減らすことなどが考えられる。また、メッセージ ID やトピック名の管理アルゴリズムにも課題がある。今回の実装ではテーブルの探索に両者共に O(logN) の時間を要する。トピック名に関しては、メッセージ頻度が多いトピック ID については探索時間が小さくなるようなアルゴリズムを用いるなど、メッセージ頻度に応じたテーブル探索アルゴリズムを選択することにより、改善可能であると考えられる。

MQTT-SN のメッセージ処理数が低下する 2 つ目の理由は、処理を行うノードがゲートウェイの分だけ HTTP や MQTT と比べ 2 つ多くなっていることが考えられる。これについては MQTT-SN のアーキテクチャ上、改善する余地がないため許容するしかない。しかし、1 つ目の理由で述べたパケット生成アルゴリズムを改善することができれば、ゲートウェイの処理が無視できる時間に収まる可能性はある。

7. まとめ

MQTT-SN のクライアントとゲートウェイを実装して MQTT、HTTP との比較評価を行った。今後の課題としてパケット生成処理、特にメッセージ ID の生成処理と、メッセージ ID とトピック名の管理アルゴリズムの改善が挙げられる。また、MQTT-SN アーキテクチャでは MQTT と比較して 2 つのゲートウェイの分だけ処理を行うノードが増えるが、メッセージ処理数を考える上でこのゲートウェイの増加による影響が無視できない問題となることが分かった。ゲートウェイに関してはこの他にスケーラビリティの問題などが考えられるが、これについても今後の課題としたい。

謝辞 本研究の一部は、JSPS 科研費 16K00147 の助成によるものである。

参考文献

[1] The MQTT protocol, http://mqtt.org（参照 2017-03-14）
[2] MQTT for Sensor Network (MQTT-SN) Protocol Specification, http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf （参照 2017-03-14）.
[3] MQTT, IBM MessageSight for Developers を使って MQTT を体験する, https://www.ibm.com/developerworks/jp/websphere/library/connectivity/ms_mqttintro/（参照 2017-04-27）
[4] 粕屋貴司, 近藤正芳, 茂手木直也, 矢野雅, 秋山貴紀, 境野哲, 貞田洋明, 堀越崇, 畠山英之, スマートシティのための MQTT プラットフォームの検証、情報科学技術フォーラム講演論文集 13(4), 1-6, 2014-08-19
[5] パブリッククラウドサービスのクラウド・エヌ,http://www.ntt.com/business/services/cloud/iaas/cloudn.html （参照 2017-03-14）
[6] Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, Hongtaek Ju, Dept. of Computer Engineering Keimyung University Daegu, Republic of Korea, Correlation Analysis of MQTT Loss and Delay According to QoS Level、ICOIN '13 Proceedings of the 2013 International Conference on Information Networking (ICOIN) Pages 714-717
[7] Power Profiling: MQTT on Android,http://stephendnicholas.com/posts/power-profiling-mqtt-on-android （参照 2017-05-08）
[8] Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android, http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https （参照 2017-05-08）
[9] Paho,https://eclipse.org/paho/ （参照 2017-05-08）
[10] Mosquitto,https://mosquitto.org/ （参照 2017-05-08）
[11] The Apache HTTP Server Project, https://httpd.apache.org/（参照 2017-04-27）

Copyright (c) 2017 by the Information Processing Society of Japan