「急速に発展するリアルタイム分散システム」特集号　　解　説

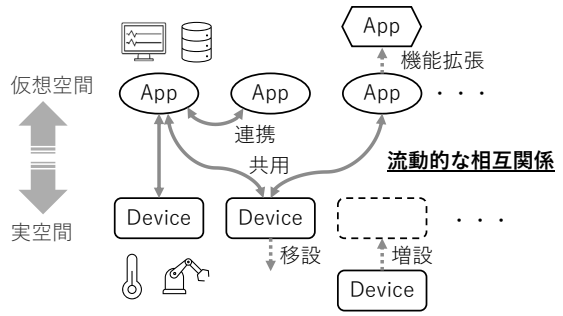
MQTT による疎結合 IoT システム

坂野　遼平*

1. はじめに

Internet of Things (IoT) という言葉が生まれてから20年以上が経つ[1]．身の回りのモノがつながる社会像は，それ以前からも Ubiquitous や Pervasive などの言葉とともに語られてきており，コンピュータとネットワークに携わる人々にとって受容しやすいイメージであったのかもしれない．近年に至り，デバイスの性能向上や通信インフラの高速化，クラウドサービスの普及が進んだことで，広範な産業分野において IoT の実用化が見られるようになってきた．たとえば製造業においては，産業用ロボットと各種センサが連携することで作業の自動化，加工時間の予測，予防保全等につなげるシステムが実現されている．

こうした IoT システムにおいて鍵となるのは，コンポーネント間の疎結合性である．IoT に通底するコンセプトはネットワークへの接続性を有するデバイスとそれらを活用するアプリケーションの連携であり，実空間と仮想空間の接続である．デバイスとアプリケーションの関係を固定的に考えることは難しく，センサデバイスの増設やアプリケーションの機能拡張といった変化は常に生じ得る．第 1 図のように多様なアプリケーション間の水平連携をも考えると，デバイスとアプリケーションの関係は一層流動的となる．デバイスとアプリケーションを相互に特化させた形で構築する密結合システムでは，そうした変化への対応に難しさが生じる．

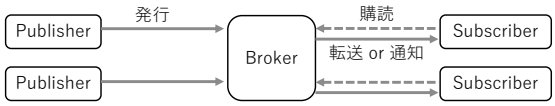


第 1 図　IoT における接続関係の流動性

* 工学院大学

Key Words: MQTT, Publish/Subscribe, IoT.

[1]1999 年に Kevin Ashton 氏（当時マサチューセッツ工科大学）が用い始めたとされている．



第 2 図　Publish/Subscribe モデル

疎結合性を得られる通信モデルとして，Publish/Subscribe モデルが挙げられる．本稿では，Publish/Subscribe モデルに基づく代表的な通信プロトコルであり，かつ IoT システムにおいて広く用いられている MQTT[1] について，特徴や研究動向などを解説する．

本稿の構成は以下のとおりである．まず **2.** 章では Publish/Subscribe モデルについて概観する．**3.** 章では MQTT の概要，応用事例，主要機能等について述べる．**4.** 章では，筆者の取り組みも含め，MQTT に関する研究動向を紹介する．最後に **5.** 章にてまとめを述べる．

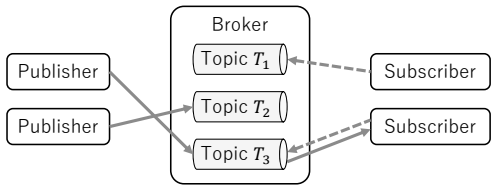
2. Publish/Subscribe モデル

Publish/Subscribe モデルは，送受信者の間に Broker（仲介者）を配置する通信モデルである．第 2 図のように，送受信者はそれぞれ Publisher（発行者）および Subscriber（購読者）とよばれ，Broker を介してやり取りを行う．Subscriber は自身が必要とする情報がどのようなものかを Broker に提示して購読を行い，Publisher は Broker に対して情報を発行する．Broker は Publisher から情報が届くと Subscriber の購読状況と照合し，その情報を必要とする Subscriber へ転送するか，あるいは情報の新着を通知する．IoT においては，たとえばセンサが Publisher となり，センサデータを分析するアプリケーションが Subscriber となる形が典型的である．

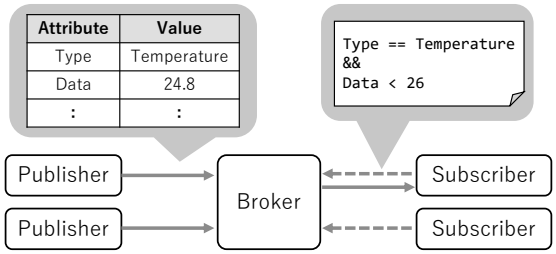
このように Broker を介することで，Publisher と Subscriber は疎結合にやり取りを行うことができる．Eugster らは，Publish/Subscribe モデルの導入により得られる疎結合性として以下の 3 種類を挙げている [2].

- 空間的疎結合性：Publisher と Subscriber はお互いを知る必要がない．
- 時間的疎結合性：Publisher と Subscriber は同時に動作する必要がない．
- 同期的疎結合性：Publisher と Subscriber はそれぞれ非同期的に通信し得る．

こうした疎結合性は，変化への柔軟な対応を可能とす



第 3 図　トピックベースの Publish/Subscribe



第 4 図　コンテンツベースの Publish/Subscribe

る．たとえば，あるアプリケーションが利用するセンサデバイスを増設した場合に，アプリケーションの設定や実装の修正を不要とする（あるいは最小限とする）ことができる．アプリケーションはあくまでも Broker と通信をしており，センサデバイスに対する直接的な通信管理は担っていないためである．このように，疎結合ゆえに Publisher はデータの生成に，Subscriber は消費に専念でき，システム構成の変化に対応しやすい点が Publish/Subscribe モデルの利点といえる．

Publish/Subscribe モデルにはいくつかの類型があり，代表的なものとしてトピックベースとコンテンツベースが挙げられる．トピックベースは，Broker における照合を「トピック」のマッチングで行う．第 3 図に示すように，Publisher は発行する情報にトピックを付与し，Subscriber も購読時にトピックを指定する．この方式に対応した代表的なプロトコルとしては，MQTT のほか，AMQP や STOMP が挙げられる．一方，コンテンツベースは Publisher が発行する情報（コンテンツ）に対して Subscriber が条件を指定する方式である．典型的には，第 4 図のようにコンテンツは複数の属性値をもち，Subscriber はそれら属性値に対する条件を指定する．トピックベースと比べ柔軟な購読が可能であるが，Broker の処理が複雑化するため実用例はやや限られる．前述の AMQP では，Headers Exchange 機能を用いることで，限定的ではあるもののコンテンツベースのやり取りが可能である．コンテンツベースの研究事例としては Meghdoot[3] や Gryphon[4] が知られている．

本稿で解説する MQTT は，前述のとおりトピックベースである．ただし **3.3** 節で述べるようにトピックの指定においてワイルドカードを利用可能であり，トピックの完全一致のみを行う Publish/Subscribe 型システムと比べると柔軟な購読が可能となっている．

なお，Publish/Subscribe 型システムの中には独立した Broker を設けない構成 (Broker-less) も存在する．たとえば DDS[5] は IP マルチキャストなどを用いてクライアントの相互発見を行い，Broker に相当するサーバを用いずにトピックベースの Publish/Subscribe モデルに基づくやり取りを実現している．クライアントの観点からは，DDS ミドルウエアが仮想的な Broker として振る舞っていると捉えることもできる．

3. MQTT の概要

MQTT はトピックベースの Publish/Subscribe モデルに基づくプロトコルである．AMQP 等の他プロトコルと比較しヘッダが小さく，処理能力や電力供給に制約のあるセンサデバイスなどとの相性がよいことから，IoT システムにおいて広く用いられている．

3.1 歴史

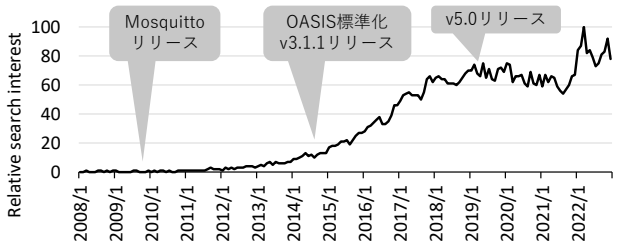
現在の MQTT の元となるプロトコルは，1990 年代後半に Andy Stanford-Clark (IBM) および Arlen Nipper（Arcom，現 Eurotech）らによって開発された．当初は衛星通信を使った石油パイプラインのモニタリングが想定されており，狭帯域かつ不安定な通信環境に耐えるよう設計されたプロプライエタリなプロトコルであった．その際に取り入れられたヘッダの軽量性や QoS 制御（後述）などの特徴が現在の MQTT にも受け継がれている．なお，初期の MQTT は幾度か名称の変遷を経ており，ArgoOTWP，MQIpdp，MQIsdp などとよばれていた[1]．

一般向けの Broker 実装としては 2008 年に IBM から Really Small Message Broker（RSMB）がクローズドソースでリリースされ，2009 年にはオープンソースの Broker 実装である Mosquitto[6] が開発・公開された．翌 2010 年，IBM と Eurotech は MQTT v3.1[7] の仕様をロイヤリティフリーで公開し，2011 年には MQTT のクライアント実装を Eclipse 財団に寄贈している．現在オープンソースの MQTT クライアント実装として広く用いられている Eclipse Paho は，この際に寄贈されたコードが元となっている．

その後，2014 年に OASIS による標準化がなされ，MQTT v3.1.1[8] がリリースされた．2016 年には ISO 標準となり，さらに 2019 年には MQTT v5.0[1] がリリースされて大幅な機能拡張がなされた[2]．オープンな仕様と実装の存在，国際標準化，そして IoT の潮流により，MQTT は 2010 年代中頃より大きな注目を集めている（第 5 図参照）．現在，MQTT Broker の主要な実装としては前述の Mosquitto に加え HiveMQ，VerneMQ，EMQX 等があり，v5.0 への対応が進みつつある．

[1] MQTT v3.1 のヘッダではプロトコル名が MQIsdp となっており，旧名称の名残が見られる．

[2] v4.x は公式には存在しない．これは，v3.1.1 のヘッダにおいて，v3.1 との区別のためにバージョン識別番号（Protocol Level）を 4 としていたためと思われる．



第 5 図 MQTT の検索トレンド（Google Trends[9] のデータから作成）

3.1.1 派生プロトコル

MQTT の派生プロトコルとして MQTT-SN[10] が挙げられる．センサネットワーク向けに軽量化した設計となっており，トランスポート層に UDP を利用可能である[1]ことや，トピックを短い別名（トピック ID）に置き換えてパケットサイズを抑えられるといった特徴がある．MQTT-SN の仕様は IBM が公開しており，前述の RSMB や MQTT-SN Tools といった無償利用可能な実装も存在している．本稿執筆時点では標準化はなされていないものの，OASIS の小委員会において MQTT v5.0 を踏まえた次期バージョンの MQTT-SN 仕様が検討されている [11].

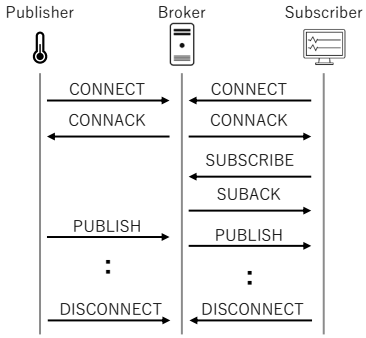
3.2 応用事例

MQTT を用いた IoT システムおよび IoT デバイスを数例挙げる．沖電気工業の monifi[12] は橋梁監視等を可能とするモニタリングサービスであり，センサデバイスとの通信に MQTT を採用している．コンシューマ市場では小型ロボットアームをはじめとするさまざまなスマートデバイスを提供している SwitchBot 社 [13] 製品等において MQTT が用いられている．IoT システムの試作などに用いられるマイコンシリーズ M5Stack でも，UART 経由で MQTT による通信を可能とする専用モジュールが提供されている [14]．また，身近なところでは Facebook Messenger において MQTT が（少なくとも当初）採用されていたことが知られている [15].

製造業においては，多様なアプリケーションの水平連携を指向した IoT 基盤の検討・開発が進んでおり，その中でも MQTT が一定の役割を担っている．Industrie 4.0 の推奨規格である OPC-UA は Publish/Subscribe 型の通信をサポートしており，MQTT および AMQP とのプロトコルマッピングが規定されている [16]．こうした状況から，Insights Hub（旧 MindSphere）[17]，FIELD System[18]，Edgecross[19] といった製造業向け IoT 基盤でも MQTT が取り入れられている．

汎用 IoT 基盤においても，通信プロトコルとして MQTT は広く採用されている．例として，IoT 基盤の国際標準規格である oneM2M[20] では，コンポーネント間の通信プロトコルとして HTTP や CoAP と並び

[1]MQTT はおもに TCP を想定している．



第 6 図 MQTT の典型的なシーケンス

主なパケット種別

	CONNECT	PUBLISH	SUBSCRIBE
固定ヘッダ	パケット種別，QoSレベル，パケットサイズ		
可変ヘッダ	プロトコル Ver.	発行トピック，パケットID	パケットID
プロパティ			
ペイロード	認証情報	任意データ	購読トピック

第 7 図 パケット構造およびおもなパケットに含まれる情報

MQTT が想定されている．また，クラウド IoT 基盤である Amazon AWS IoT Core[21] においてもクラウドとの通信に MQTT を用いることができる．このほか，Akamai IoT Edge Connect[22] では広域に配置されたエッジサーバを分散型 Broker として用い，大規模な MQTT 通信を行うことが可能となっている．

3.3 主要機能

本節では MQTT の特徴的な仕様・機能について紹介する．紙幅の都合上，網羅的な紹介は難しいため，詳細については仕様書 [1,8] を参照されたい．なお，以降では特記のない限り MQTT v5.0 を対象として述べる．

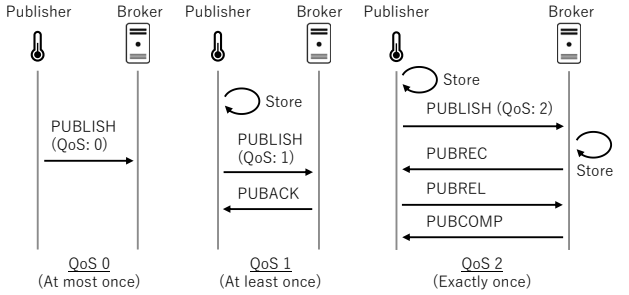
クライアントはまず Broker に CONNECT パケットを送信し，アプリケーション層のセッションを形成する（第 6 図）．そして DISCONNECT パケットによりセッションを閉じるまで，メッセージの発行や購読を継続的に行う．

MQTT パケットは，第 7 図に示すように固定ヘッダ，可変ヘッダ，ペイロードから成る．固定ヘッダはすべての MQTT パケットに必ず存在し，CONNECT や PUBLISH といったパケット種別の指定，パケットサイズに関する情報などを含む．なお，MQTT パケットのサイズ上限は約 268MB である．可変ヘッダおよびペイロードの有無は，パケット種別により異なる．たとえば，死活監視に用いられる PINGREQ パケットは固定ヘッダのみから構成され，可変ヘッダとペイロードは存在しない．センサデータなどの送信に用いられるパケット種別は PUBLISH であり，そのヘッダサイズは最小で 6 バイトである[2].

[2]固定ヘッダが 2 バイト，可変ヘッダがトピック名 3 バイトおよびプロパティ長 1 バイトの場合．トピック名が 1 文字といったやや極端なケースである．

第 1 表　ワイルドカードによるトピック指定の例

購読トピック	マッチする発行トピックの例
foo/bar/#	foo/bar, foo/bar/baz, foo/bar/qux/baz
foo/+/baz	foo/bar/baz, foo/qux/baz



第 8 図　QoS 制御

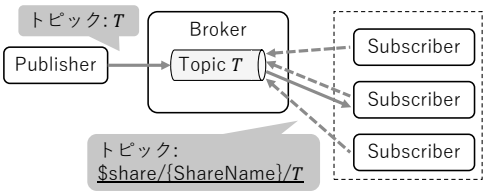
MQTT におけるトピックはスラッシュ区切りの階層となっている．たとえば “japan/tokyo/temperature” や “japan/tokyo/humidity” といった形でトピックを指定できる．Subscriber は，購読トピックの指定時に 2 種類のワイルドカードを利用することが可能である．マルチレベルワイルドカード “#” は，当該階層以下の全階層について任意の文字列とマッチする．シングルレベルワイルドカード “+” は，当該 1 階層についてのみ任意の文字列とマッチする．第 1 表に例を示す．なお，Publisher はワイルドカードを用いることはできない．

MQTT が v3.1.1 以前から備えている特徴的な機能として，QoS レベルの指定が挙げられる．これはメッセージの到達確認と再送の仕組みである．第 8 図に示すように，0，1，2 の 3 段階のレベルが指定でき，送信に対する応答（PUBACK，PUBREC，PUBREL，PUBCOMP）が届かない場合には再送を行うことで，それぞれメッセージが最大 1 回，最低 1 回，そして過不足なく 1 回届く形となる．なお，QoS 制御は Publisher と Broker，Broker と Subscriber の間でそれぞれ行われるものであり，Publisher と Subscriber 間の到達を直接保証するものではない．

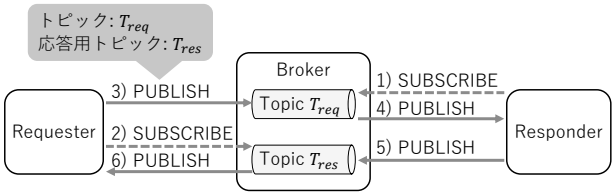
3.3.1　MQTT v5.0 における機能拡張

MQTT v5.0 から新たに導入された機能としては，Shared subscription，Reqeust/response，Server redirection が挙げられる．いずれも，MQTT v5.0 で可変ヘッダ内に新たに設けられたプロパティ領域を活用して実現されている．

Shared subscription は複数 Subscriber でグループを形成し受信を分担する機能である．第 9 図に概略を示す．Subscriber は，購読時のトピックに “$share” から始まるプレフィクスを付ける．“ShareName” はグループ名に相当し，同じ ShareName を指定した Subscriber 同士がグループを形成する．当該トピック（図の例では T）に発行されたメッセージは，グループ内のいずれかの Subscriber のみが受信する．その際，どの Subscriber



第 9 図　Shared subscription の概略



第 10 図　Request/response の典型的な流れ

が選ばれるかは Broker の実装依存である．

Reqeust/response は，クライアント間で要求と応答を交わすことを可能とする仕組みである．第 10 図に示すように，要求側クライアントは要求メッセージを発行する．その際，可変ヘッダのプロパティ領域にて応答用トピックを指定する．当該メッセージを受信したクライアントは，応答用トピックに対して応答メッセージを発行する．要求側クライアントは事前に応答用トピックを購読しておくことで，応答を受け取ることが可能となる．

Server redirection は，Broker からクライアントに対してほかの接続先 Broker を提示するための仕様である．クライアントからの接続要求（CONNECT パケット）に対する応答である CONNACK パケットに，ほかの接続先 Broker の情報を載せることができる．また，Broker からクライアントへの切断通知（DISCONNECT パケット）にも同様の情報を載せることが可能である．これらにより，クライアントに他 Broker への接続を促すことが可能となっている．

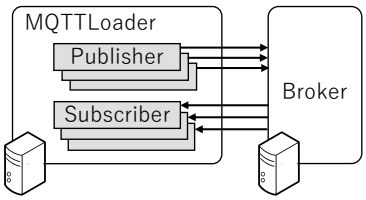
本節で述べてきた諸機能を検証できる場として，複数のパブリック Broker が提供されている [23]．Mosquitto や HiveMQ など，v5.0 に対応したものもあり，さまざまな機能を手軽に試すことが可能である．

4.　課題と研究動向

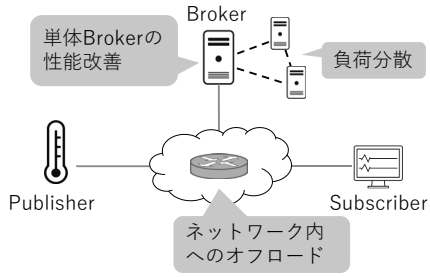
MQTT の標準的なアーキテクチャでは Broker がすべてのメッセージを処理することから，Broker の処理性能がしばしば取り沙汰される．本章では Broker 性能に関する研究動向について簡単に紹介する．

4.1　Broker 性能の測定・評価

前述のとおり，Broker の処理性能は技術者や研究者の大きな関心事項となっており，性能評価に関する数多くの研究報告がある．一例として，Bender らは MQTT のオープンソース実装である Mosquitto，HiveMQ，EMQX，VerneMQ，MQTT.js および Paho の性能評価を行っている [24]．また，Gemirter らはスマートシティの実デー



第 11 図 MQTTLoader の概略



第 12 図 Broker 性能向上へのアプローチ

タセットを用いて MQTT，AMQP，HTTP の比較評価を実施している [25].

性能測定のためのさまざまなツールも公開されている．よく知られたものとしては，eMQTT-Bench[26], mqtt-benchmark[27]，MQTT Stresser[28] などが挙げられる．われわれの研究グループでもオープンソースの測定ツールとして MQTTLoader を開発・公開している [29,30]．第 11 図のように，複数のクライアントを立ち上げて Broker に対し負荷をかけ，スループットや遅延を測定するものである．MQTT v3.1.1 および v5.0 に対応しており[1]，**3.3.1** 節に述べた Shared subscription 等を用いた性能評価も可能となっている．

4.2 Broker 性能の改善

Broker の処理性能を向上させるアプローチには，大きく三つの方向性がある．Broker 単体性能の向上，複数 Broker を用いた負荷分散，そしてネットワーク内へのオフロードである（第 12 図）.

単体性能向上の研究事例としては，muMQ[31] が挙げられる．マルチコアの活用および DPDK を用いたカーネルオーバヘッドの回避により性能向上を実現している.

負荷分散の取り組みとしては，複数 Broker を相互接続し配送木を形成する手法を提案した MQTT-ST[32] や，分散型 Broker において Broker 間の通信量を削減する手法を提案した Detti らの研究 [33] が挙げられる．また，分散型 Broker の商用サービスとしては **3.2** 節に述べた Akamai IoT Edge Connect がある．われわれも複数 Broker の連携に取り組んできており，任意の Broker 実装を相互接続可能とする ILDM[34] や，Shared subscription および Server redirection を用いたスケーラブルな IoT データ収集機構 [35,36] 等の研究を進めている.

[1] MQTTLoader 公開（2020 年 8 月）以前は，上述の既存ツールは MQTT v5.0 に非対応であった.

やや毛色の異なるアプローチとして，Broker における処理をネットワーク内へオフロードする取り組みも存在する．われわれは MQTT-SN[10] を対象に，データプレーン向けプログラミング言語 P4[37] を用いて，ネットワーク機器において Broker の一部処理を代行する技術の研究に取り組んでいる．P4 を用いて MQTT パケットを扱う取り組みとしては Ali らの SDFog-Mesh[38] も挙げられる．こうしたネットワーク内処理については制約も多いものの，Broker が担う処理の配置構成について従来より広い選択肢を与え得るものであり，今後のさらなる研究に期待したい.

5. おわりに

本稿では，IoT の主要通信プロトコルとして知られる MQTT について，通信モデルとしての位置付けと応用事例，最新仕様における機能拡張，Broker の性能観点での研究動向等を紹介した.

MQTT が備える，Publish/Subscribe モデルに由来する疎結合性は，多様なデバイスとアプリケーションの連携を図る IoT において重要な役割を果たしている．国際標準化されており，オープンソースを含め数多くの実装があること，またコンシューマ向け/エンタープライズ向けを問わず実用化実績が豊富であることから，今後も幅広い分野において活用されると考えられる．一方，今後の活用の中で課題となり得る点としてセキュリティ面が挙げられる．MQTT は TLS による暗号化通信が可能であるが，必然的にスループットや遅延への影響が生じる [39]．また，CONNECT パケットや AUTH パケットにより認証情報をやり取り可能となっているものの，詳細については適用先ごとに設計する必要がある．IoT 社会の重要な要素技術として，安全性と性能の両立や認証認可の運用に関するさらなる共有知の形成が望まれる.

謝 辞

本稿で解説した研究の一部は JST さきがけ JPMJPR21P8 の支援を受けたものである.

(2023 年 5 月 30 日受付)

参 考 文 献

[1] OASIS Message Queuing Telemetry Transport (MQTT) TC: MQTT Version 5.0 (2019)
[2] P. Th. Eugster, et al.: The many faces of publish/subscribe; *ACM Computing Surveys*, Vol. 35, No. 2, pp. 114–131 (2003)
[3] A. Gupta, et al.: Meghdoot: content-based publish/subscribe over P2P networks; *Middleware*, pp. 254–273 (2004)
[4] R. Strom, et al.: Gryphon: An information flow based approach to message brokering; *ISSRE* (1998)
[5] Object Management Group: OMG Data Distribution Service (DDS) Version 1.4 (2015)

[6] R. A. Light: Mosquitto: server and client implementation of the MQTT protocol; *Journal of Open Source Software*, Vol. 2, No. 13, p. 265 (2017)
[7] IBM, Eurotech: MQTT V3.1 Protocol Specification (2010)
[8] OASIS Message Queuing Telemetry Transport (MQTT) TC: MQTT Version 3.1.1 Errata 01 (2015)
[9] Google Trends; `https://trends.google.co.jp/trends/` (2023年5月23日参照)
[10] A. Stanford-Clark, et al.: MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2 (2013)
[11] OASIS MQTT SN Subcommittee: `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt-sn` (2023年5月23日参照)
[12] 沖電気工業株式会社: インフラモニタリングサービス「monifi」; `https://www.oki.com/jp/infra_monitoring/monifi/` (2023年5月23日参照)
[13] SwitchBot; `https://www.switchbot.jp/` (2023年5月23日参照)
[14] M5Stack: UNIT MQTT PoE; `http://docs.m5stack.com/en/unit/mqtt_poe` (2023年5月23日参照)
[15] Meta: Building Facebook Messenger; `https://engineering.fb.com/2011/08/12/android/building-facebook-messenger/` (2023年5月23日参照)
[16] OPC Foundation: OPC 10000-14: UA Part 14: PubSub v1.05.02; (2022)
[17] Insights Hub; `https://documentation.mindsphere.io/` (2023年5月23日参照)
[18] FIELD System; `https://www.fanuc.co.jp/ja/product/field/` (2023年5月23日参照)
[19] Edgecross; `https://www.edgecross.org/` (2023年5月23日参照)
[20] oneM2M; `https://www.onem2m.org/` (2023年5月23日参照)
[21] Amazon AWS IoT Core; `https://aws.amazon.com/jp/iot-core/` (2023年5月23日参照)
[22] Akamai IoT Edge Connect; `https://techdocs.akamai.com/iot-edge-connect/docs` (2023年5月23日参照)
[23] mqtt.org: public_brokers; `https://github.com/mqtt/mqtt.org/wiki/public_brokers` (2023年5月23日参照)
[24] M. Bender, et al.: Open-source MQTT evaluation; *IEEE CCNC*, pp. 1–4 (2021)
[25] C. B. Gemirter, et al.: A comparative evaluation of AMQP, MQTT and HTTP protocols using real-time public smart city data; *International Conference on Computer Science and Engineering*, pp. 542–547 (2021)
[26] eMQTT-Bench; `https://github.com/emqx/emqtt-bench` (2023年5月23日参照)
[27] mqtt-benchmark; `https://github.com/krylovsk/mqtt-benchmark` (2023年5月23日参照)
[28] MQTT Stresser; `https://github.com/inovex/mqtt-stresser` (2023年5月23日参照)
[29] MQTTLoader; `https://github.com/dist-sys/mqttloader` (2023年5月23日参照)
[30] R. Banno, et al.: Measuring performance of MQTT v5.0 brokers with MQTT loader; *IEEE CCNC*, pp. 1–2 (2021)
[31] W. Pipatsakulroj, et al.: muMQ: A lightweight and scalable MQTT broker; *IEEE LANMAN*, pp. 1–6 (2017)
[32] E. Longo, et al.: MQTT-ST: A spanning tree protocol for distributed MQTT brokers; *IEEE ICC*, pp. 1–6 (2020)
[33] A. Detti, et al.: Sub-linear scalability of MQTT clusters topic-based publish-subscribe applications; *IEEE Transactions on Network and Service Management*, Vol. 17, No. 3, pp. 1954–1968 (2020)
[34] R. Banno, et al.: Interworking layer of distributed MQTT brokers; *IEICE Transactions on Information and Systems*, Vol. E102.D, No. 12, pp. 2281–2294 (2019)
[35] R. Banno, et al.: A scalable IoT data collection method by shared-subscription with distributed MQTT brokers; *EAI MONAMI*, pp. 218–226 (2022)
[36] 吉村ほか: Server redirectionを利用したMQTTブローカへのクライアント振り分け手法; 電子情報通信学会総合大会, p. B-16-10 (2023)
[37] P. Bosshart, et al.: P4: Programming protocol-independent packet processors; *ACM SIGCOMM Computer Communication Review*, Vol. 44, No. 3, pp. 87–95 (2014)
[38] S. Ali, et al.: SDFog-Mesh: A software-defined fog computing architecture over wireless mesh networks for semi-permanent smart environments; *Computer Networks*, Vol. 211, p. 108985 (2022)
[39] V. Seoane, et al.: Performance evaluation of CoAP and MQTT with security support for IoT environments; *Computer Networks*, Vol. 197, p. 108338 (2021)

著者略歴

坂野（ばんの） 遼平（りょうへい）

2010年北海道大学工学部情報エレクトロニクス学科卒業，2012年同大学大学院情報科学研究科修士課程修了，2018年東京工業大学大学院情報理工学研究科博士課程修了．2012年日本電信電話株式会社入社，2018年東京工業大学情報理工学院研究員，2020年工学院大学情報学部情報通信工学科助教，2022年より准教授，現在に至る．IoTシステム，オーバレイネットワークなどの研究に従事．博士（理学）（2018年9月，東京工業大学）．2015年度情報処理学会論文賞，2019年度船井研究奨励賞などを受賞．ACM，IEEE，IEICE，IPSJ各会員．