共同研究報告書(2023 年度)

研究課題: MQTT 上に実装した RPC プロトコルを用いる IoT 応用サービスの研究

研究代表者　北海道大学大学院情報科学研究院
教授　土橋 宜典

あらまし 我々は多数のエッジノードが連携して実現する IoT サービスの基本通信プロトコルとして MQTT を下層プロトコルに用いる RPC(Remote Procedure Call)として MQTTRPC の実装を進めている。これまでの研究成果として汎用 IoT エッジノード Raspberry Pi 上に MQTTRPC の実装を完了し、IoT カメラなどのアプリケーション動作を確認できている。その成果を踏まえ、2023 年度は MQTTRPC を応用する具体的サービスを開発し、IoT サービスとしての実用性の検証と運用技術の開発を目的として本研究を実施した。具体的な成果として、MQTTRPC を用いて北海道大学が運行する構内循環バスのリアルタイムバスロケーションサービスを実装し、MQTTRPC が実用サービスに応用可能であることを示すことができた。また、大規模なデータ流通が必要な応用として、IoT カメラを用いた人流計測のプロトタイプを開発し、MQTTRPC が大量データ通信を必要とするサービスに適用可能であることを確認した。

キーワード　Society 5.0, IoT, MQTT, RPC, ブローカー, バスロケーション、スリットカメラ、人流計測

1. まえがき

サイバー・フィジカルシステムは物理世界と情報世界をネットワークで融合し、効率改善や新サービスの創出につながると言われている。サイバー・フィジカルシステムは物理空間に配置されネットワーク接続された複数の小型コンピュータ（エッジノード）、サーバー、ユーザー端末(クライアント)で構成される大規模分散システムである。近年の半導体技術の革新にエッジノードの高機能化、大容量化が進み現在ではフルスペックの UNIX 準拠オペレーションシステムが搭載されるようになっている。。

また、IoT 分野ではネットワークセキュリティ上の問題からエッジノードにグローバル IP アドレスを割り当てず、外部からの接続ができないローカルアドレスによる接続とすることが求められる。用途によっては常時接続ではなく、間欠的なネットワーク接続により消費電力や通信コストを低減することも求められる。このように制限されたネットワーク環境で、ローカルアドレスしか持たないノード間で遠隔プログラム実行（RPC-Remote Procedure Call）を実現するのは簡単ではない。

我々の研究グループでは、IoT 分野での応用を前提とした RPC プロトコルとして MQTT プロトコルを下層プロトコルとして用いる RPC サービスである MQTTRPC の実装を進めてきた。 MQTT はオーバーヘッドの少ない軽量プロトコルであるが、単純なテキストメッセージの交換しか提供されず、高機能なサービスに使う場合はさらに上層のサービスプロトコルを設計する必要が出てくる。その上位プロトコルの共通部分を提供するのが MQTTRPC である。

2. MQTTRPC の通信モデルと基本プロトコル

2.1. MQTTRPC の構成要素

MQTTRPC は MQTT を下層プロトコルとして、その上に実装する遠隔プログラム実行プロトコルである。MQTTRPC の構成要素は以下に示す 3 要素からなる。

1. MQTT Broker
2. Edge Node
3. Client Node

MQTT Broker は各ノードがメッセージ交換するときに使用する中継機能を提供する。実態は標準的な MQTT Broker (Mosquito)である。Edge Node と Client Node は動作上の違いは無いが、動作環境が異なっている。Edge Node は IoT サービスではセンサーのように遠隔地の無人環境で動作するノードコンピュータを想定している。一方、Client Node は人間が操作するブラウザ画面をイメージしてい

る。クライアントノードはブラウザのように、人間が操作する時だけ一時的に動作するものを想定している。これら 3 要素の関係を図 1 に示す。

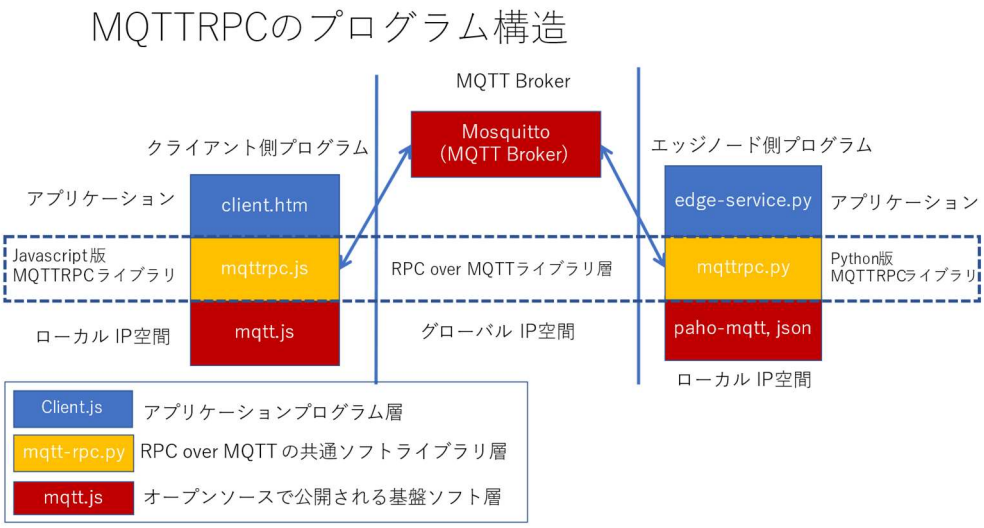

図 1. MQTTRPC の構成要素

2.2. MQTTRPC のメッセージ交換パターン

MQTTRPC はエッジノード上では常駐プロセス（MQTTRPC Daemon）として実行されている。ノード上の MQTTRPC Daemon は重要な動作パラメータとして以下の 4 パラメータを使う。

1. MQTT Broker URI
2. Service Key
3. Service Class
4. Node ID

MQTT Broker URI は各ノードが RPC メッセージを交換する時に経由する MQTT Broker のアクセス情報であり、IP アドレスと TCP ポート番号で定義される。Service Key は MQTT Broker が管理するサービス空間を識別するものである。Service Key を共有するノードに対しては同報通信を行うことができる。Service Key は Major Key と Minor Key の 2 階層で記述される。Major Key やサービス提供者の名称など大枠でのくくりであり、個々のサービス名は Minor Key で指定することを想定している。つまり、一つの Major Key に対して、複数の Minor Key を定義できる。Service Class は Service Key で指定されたサービス空間に存在するノードの種類(Class)を指定する。一般的な応用では”edge”と”client”の 2 種類のクラスが使用される。Node ID は使用する MQTT Broker 内でユニークな ID でノードの識別のために使われる。Node ID は MAC アドレスや CPU ID のようにユニーク性が担保されたものを使うことができる。

2.3. MQTTRPC のトピックパターン

MQTTRPC では MQTT の TOPIC パターンに通信の相手先、通信モードを記述し、メッセージ部に RPC オブジェクトを JSON 形式で記述する。MQTTRPC が発する RPC リクエストは以下の 5 階層の

TOPIC パターンを持つ。

major_key / *minor_key* / *service_class* / *dst_node* / *src_node*

各階層の値（文字列）は'*'を使用することでワイルドカードとして機能させることができる。

例えば、*service_key* = it-prototyping/camera のサービスに含まれる、*src_id* = dc1234567 から *dst_id* = b81234567 に対して RPC メッセージを送る場合は、以下のトピックパターンになる。

it-prototyping / camera / * / b81234567 / dc1234567

これは、１対１の通信パターンである。

もう一つのパターンとして、*service_class* に含まれるノードすべてに同報型の RPC メッセージを送る TOPIC パターンがある。src_id = dc1234567 から service_class = edge であるノードすべてに RPC メッセージを送る場合は以下のトピックパターンになる。

it-prototyping / camera / edge / * / dc1234567

これは service_key と serivice_class が同一である全ノードに対してメッセージを送る通信パターンである。図 2-1, 図 2-2 に通信パターンを図示する。

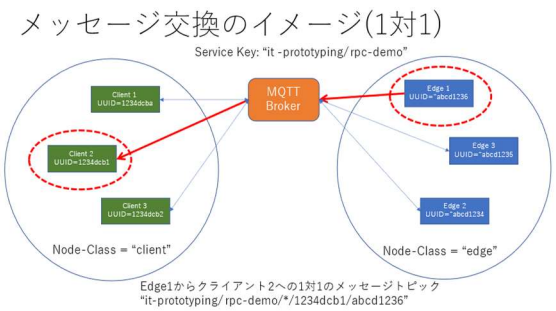

図 2-1 Node ID 指定の通信パターン

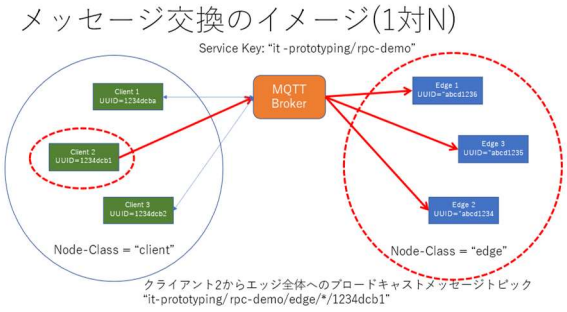

図 2-2 Service Class 指定の通信パターン

2.4. MQTTRPC が提供する基本機能

MQTTRPC では、ノードは TOPIC パターンで指定するノードに対して、RPC メッセージを送り、必要に応じて値を得ることができる。RPC メッセージは JSON 形式で定義される。MQTTRPC が提供する機能は以下のように説明される。

1. MQTTRPC を発するノード(Source Node)から指定したノード(Destination Node)へ JSONRPC2.0 に準拠した形式で RPC メッセージを送出し、結果を受け取る
2. MQTTRPC の基本ライブラリはメソッドとして Ping, Get, Put, Signal, Shell を基本組み込みメソッドとして内蔵しており、これだけである程度の応用プログラムを作成できる。
3. 新規にメソッドを定義する仕組みを提供する。
4. 複数ノードに対してブロードキャストリクエストを送付する仕組みとして応答を待たない特殊形式の Signal メソッドを提供する。

これらの機能を実現するために、MQTTRPC では MQTT のペイロード部に記述される JSON 形式のオブジェクト形式を規定している。

3. バスロケーションサービスへの MQTTRPC の応用

MQTTRPC を用いたサービスとして、北海道大学構内循環バスのリアルタイムバスロケーションシステムを開発し、2023 年度に実際にバスに IoT 端末を搭載し社会実験を実施した。本実験では、バス 2 台に GPS および 4G/LTE 通信モジュールを搭載した座標センサー端末を開発した。バスは北大構内とは言え、実際に乗客を乗せて運行されているものであり、安全性確保のため乗務員には一切の操作を要求しないという条件を設定した。開発した GPS センサー端末は Raspberry Pi ZERO＋Linux OS で動作するが、バスではエンジン停止と同時に電源が切れるため Unix のシャットダウン動作の時間が確保できない問題がある。また、電源投入から Linux が立ち上がり、サービスが開始するまでに分単位の時間を要することや GPS が電源投入から位置を確定するまでの時間も分単位でかかることなどから、電源マネジメントはバス運行パターンや短時間のエンジンオフを考慮する必要がある。そのため、端末内に 30 分程度のバッテリバックアップ機能を組み込み現実のバス運行に耐える設計にしている。MQTT 通信には 4G/LTE の通信モジュール(USB ドングル)を用いている。

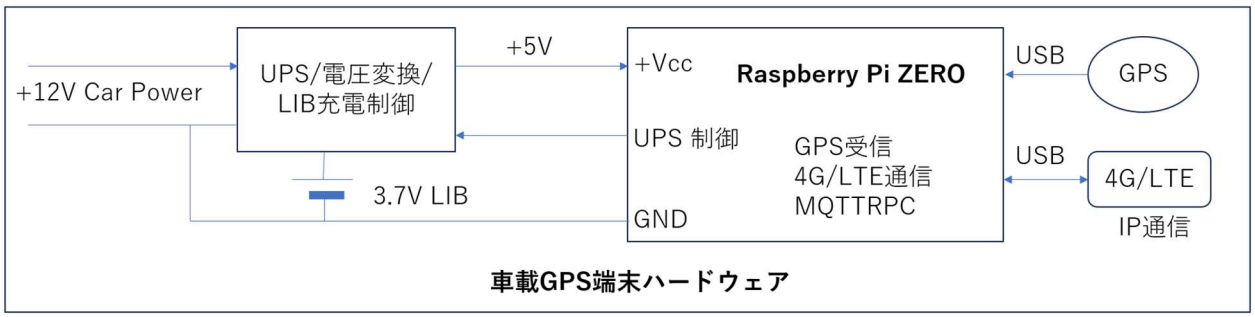

図 3.1 車載 GPS 端末のブロック図

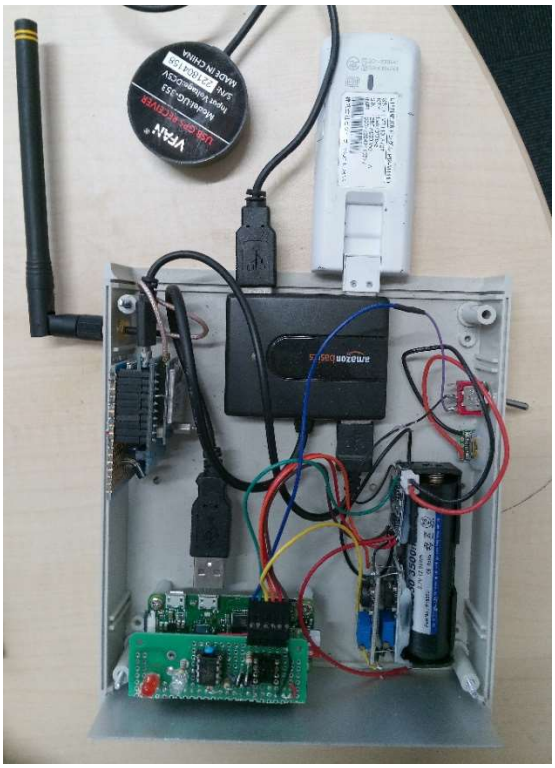

図 3.2 車載 GPS 端末の実装

本バスロケーションサービスの実装では MQTRPC 層では以下の3種のノードクラスを用いている。

1. Edge class (minor key = edge)
   車載 GPS 端末のクラス、 通信対象は repeater class
2. Repeater class (minor key = repeater)
   中継ハブのクラス、通信対象は edge, client
3. Client class (minor key = client)
   ブラウザ端末のクラス、通信対象は repeater

各ノードでは常駐プログラムの内部で MQTTRPC のモジュールが実行されており、自分に向けた MQTTRPC メッセージを受信した場合はそれに対する処理を実行する。MQTTRPC の立ち上げ段階で、下層プロトコルとして MQTT ブローカーとの接続を行っている、つまり、MQTTRPC からは見えないが、実際には MQTT レベルのメッセージを交換するためのブローカーサービスが存在している。MQTT と MQTTRPC の関係を含めた、バスロケーションサービスの関係を図 3.3 に、ブラウザ上の表示画面例を図 3.4 に示す。

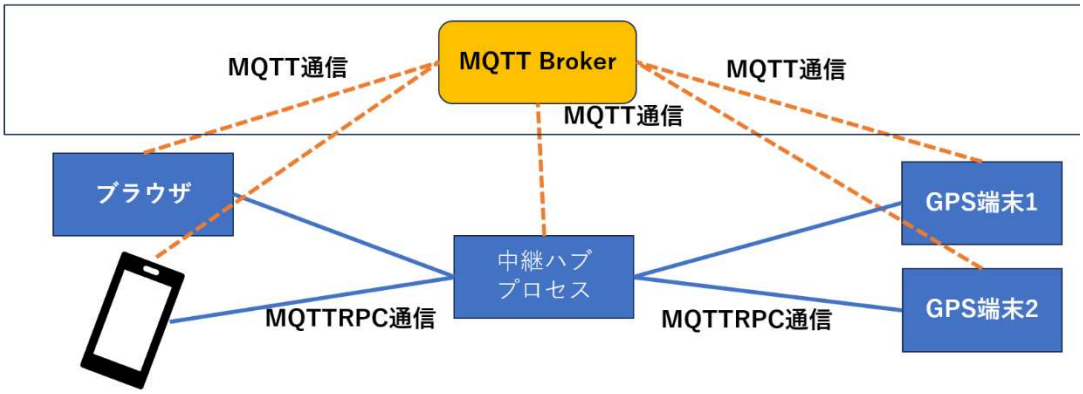

図 3.3 MQTTROC によるバスロケーションサービスの実装モデル

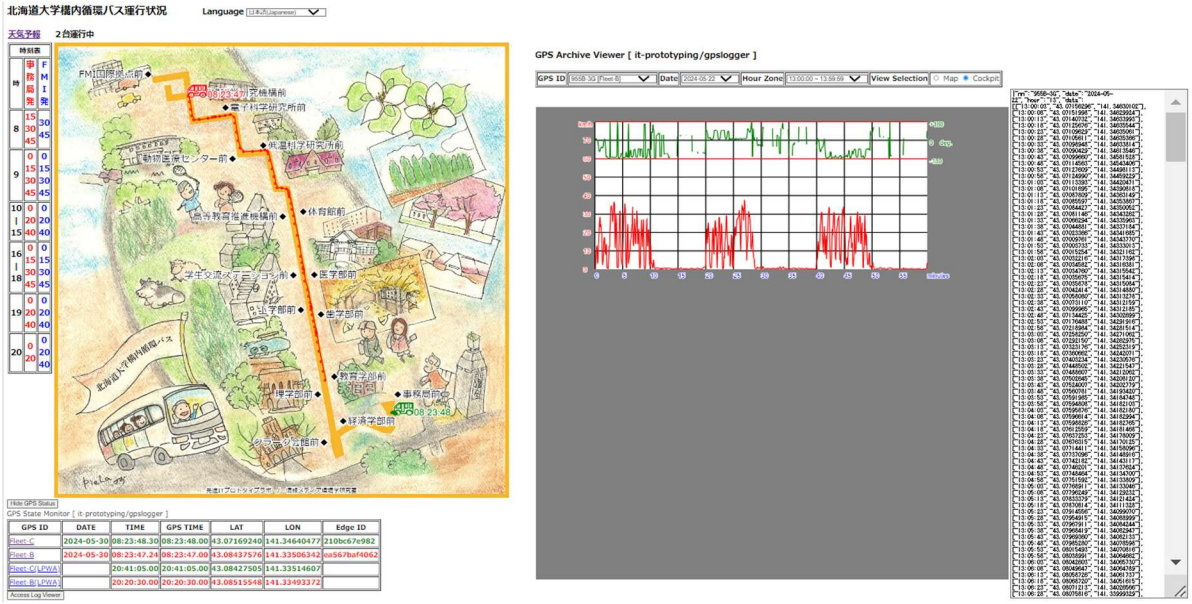

図 3.4 サービス画面

4. スリットカメラによる人流計測サービスへの応用

MQTT はテキストベースの小規模データの交換を意図して設計、実装されており画像や音声など大規模かつリアルタイム性が求められる応用には適さないとされている。しかし、仕様上は最大メッセージサイズが 256MB となっており、高解像画像でも交換可能な印象を受ける。実際は MQTT ブローカの実装により最大メッセージサイズが制限されているが、通常は 256KB 程度のデータ交換は可能である。256KB は非圧縮 8bit モノクロ画像では 1024×256 画素程度であり、JPEG 圧縮を前提とすれば 1024×1024 画素程度のモノクロ画像の交換が可能なレベルのパケットサイズである。高品質のカラー画像の交換には不足するパケットサイズであるが、IoT エッジ側で前処理を行った中間画像形式のデータであれば MQTT あるいは本研究で開発した MQTTRPC でも交換可能になると考えられる。

我々の研究グループでは、道路や通路などを通行する人や車両の数を正確にカウントする手法としてスリットカメラによる計測法を開発している。その人流センサーを遠隔地に設置して定常的に観測する場合に安定な通信モデルが必要になる。我々はその試作端末の通信プロトコルとして MQTTRPC の応用を検討した。

人流計測を目的としたスリット画像では人の歩行速度などを考慮するとスリットのサンプリング間隔は 1/16 秒(毎秒 16 ライン)程度で十分である。その場合、1分間のスリット画像の時間方向画素数(X 軸)は 16×60、つまり 960 画素となる。Y 方向画素数は観測対象によるが、人流計測であれば数百画素で十分である。我々の実験環境では、カメラ視野内に 2 列のスリットを定義することにより、カウント対象が左右どちらの方向に移動しているかを判断可能とすることを想定しており、1スリット当たり Y 軸方向に 180 画素でスリット画像を構成している。つまり、Y 軸は 180×2 列 = 360 画素/Line であるので、1分間のスリット画像データは 360×960 画素で構成されることになる。図 4.1 にその条件で撮影した1分間のスリット画像の例を示す

図 4.1 道路上の人流スリット画像の例(180 画素×2 スリット, 60 秒)

この例では、目視でも 1 分間に約 75 人の人流があったことが読み取れる。この画像をパターン認識処理により人流を自動カウントすることが求められるが、その処理には相当量の計算処理が必要となる。

この画像を JPEG 圧縮した結果、最終的には 130KB のバイナリデータとして表現されることが確認で

きた。つまり、このレベルに前処理された画像データであれば MQTTRPC のペイロードに乗せることができることが確認できた。

5. むすび

MQTT 上に実装した RPC プロトコルである MQTTRPC とその応用例の開発について報告した。MQTT は低機能でシンプルな通信モデルであるが、不安定な通信環境でも安定して動作する特徴を備えている。本研究で開発した応用例はその特徴を生かしたものであり、MQTTRPC で実用レベルのサービスが実現できることを示すことができた。

また、現状でもエッジとサーバーで処理分担することで画像や音声といった大量データを MQTT により交換するサービスも実現可能であることを示すことができているが、今後MQTTの実装水準が上がってくればより大量のデータの取り扱いが可能になり、応用範囲が広がるであろう。

現在の実装ではエッジ側も Linux レベルの OS で動作している前提であるが、RTOS レベルの低水準OSでもMQTTRPCが道さできるように、MQTTROCの実装も強化する必要がある。