共同研究報告書(2022 年度)

研究課題: IoT サービス用 RPC プロトコル(MQTTRPC)の実装

研究担当者　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 Society 5.0 社会の実現に向けて、DX(デジタルトランスフォーメーション)が各分野で加速している。特にサイバー・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大量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り、その安定動作の実現は容易ではない。本研究では大規模分散システムとして実現される IoT 環境のための基本通信プロトコルとして MQTTRPC を提案し、その仕様設計及び実装を行った。本研究で開発した MQTTRPC は IoT ノード間の通信を RPC(Remote Procedure Call)のライブラリとして提供するものであり、IoT ノード間を MQTT ブローカーのみを経由して RPC による通信を可能にした。開発した MQTTRPC を用いて、遠隔地に設置されたローカルアドレスのみを持つエッジノードに監視カメラ機能を持たせ、その動作をリモート制御するサービスを開発した。

キーワード　Society 5.0, IoT, MQTT, RPC, Broker, Serverless, リモートログイン

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。

IoT 分野では単純な構造で高速なリアルタイム応答を実現するプロトコルとして MQTT が知られている。MQTT の特徴として不安定な通信環境においても安定したメッセージ交換を提供する強靭さがある。しかし、MQTT が提供するのは単純なテキストメッセージの交換のみであり、RPC で必要になるプログラムの遠隔実行やオブジェクトデータの交換などの高レベル機能は提供されない。

私たちは、実績のある MQTT プロトコルを用いて、その上層プロトコルとして汎用性の高い RPC プロトコルとして MQTTRPC を実装した。

2. MQTTRPC の通信モデル

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。

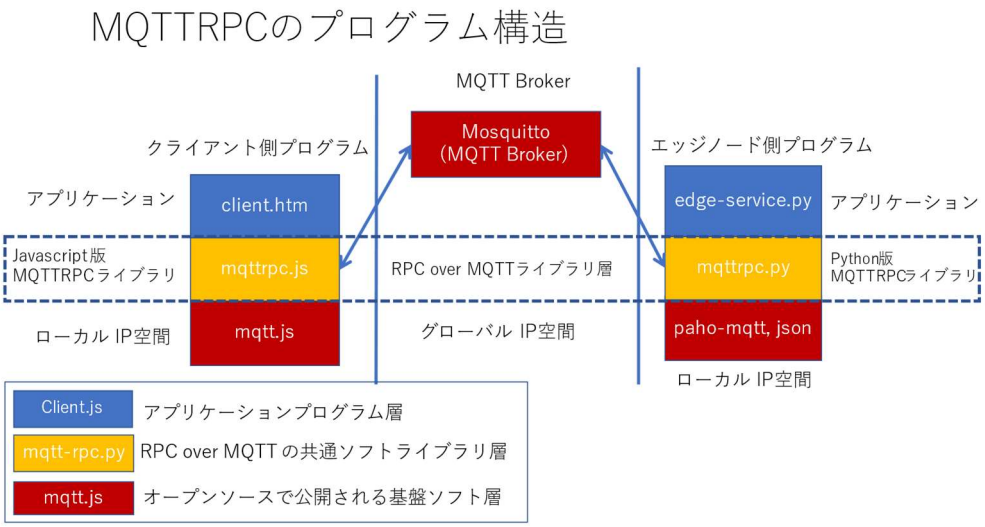

図 1. MQTTRPC の構成要素

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。

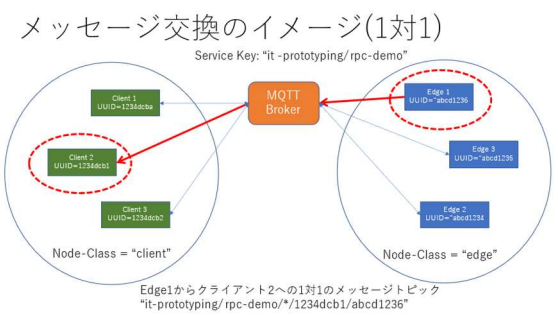

図 2-1 Node ID 指定の通信パターン

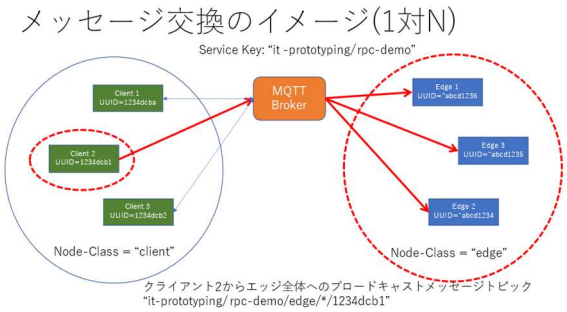

図 2-2 Service Class 指定の通信パターン

3. MQTTRPC の基本機能。

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。MQTT のペイロードにに RPC メッセージを載せる規格は MQTTRPC 2.0 として提案されている。我々の MQTTRPC はそれに準拠するが、いくつかの拡張を行っている。

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。

我々の実装において、特殊なメソッドとして Signal を定義している。Signal Method は送付先のノードに対してプログラムの実行をリクエストするが、戻り値を返さない。これは同報型の RPC に対する対応である。Signal メソッド自体は値を戻さないが、必要な場合は Signal メソッドで非同期に値を返えすことができる。

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。図 4 で MQTTRPC のペイロード上でのメッセージフォーマットの形式を解説する。

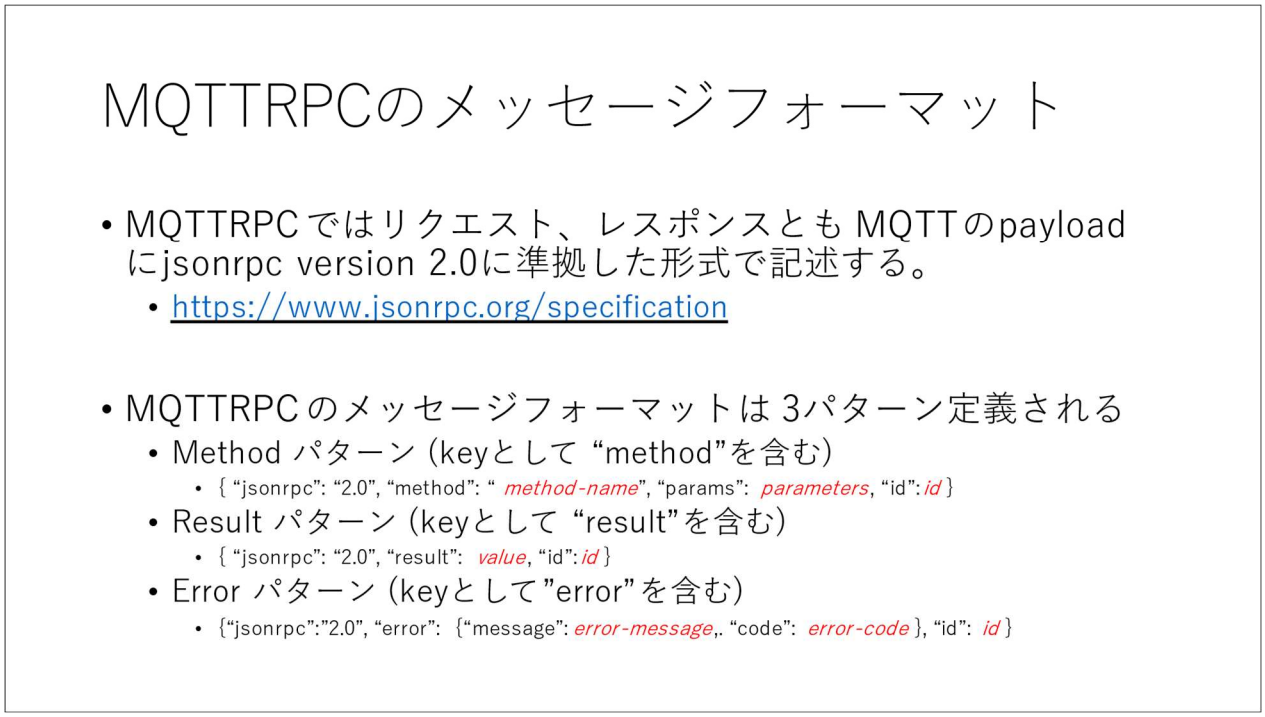

図 4. MQTTRPC のメッセージフォーマット

4. MQTTRPC の応用例

我々が実装した MQTTRPC を用いて応用システムを開発した。図 4 は北海道大学内で運行されている構内循環バスのバスロケーションサービスへの応用例である。このサービスでは、サービスクラスとして edge, repeater, clielt の 3 クラスを定義している。edge クラスはバスに搭載された GPS センサー付きエッジノードであり、５秒間隔で repeater クラスに現在位置を MQTTRPC で送信している。repeater クラスは edge クラスからのメッセージを受け取り、内部 DB にバス運行履歴を記録すると同時に、client クラスからのリクエストに対して、DB の内容を JSON オブジェクトの形式で返す。client クラスは利用者が使用するブラウザ内のアプリケーションであり、repeater が返すバスの位置情報を受け取り、可視化する処理を行う。バスに搭載するエッジノードはバス運行時しか電源が投入されないなど、不安定な通信環境であるが、MQTTRPC ではネットワークが接続と切断を繰り返すような環境でも安定に動作す

ることが確認できた。これは MQTT が不安定な環境を想定してプロトコルの実装を行っていることによるところが大きい。図５に構内循環バスロケーションサービスの画面を示す。

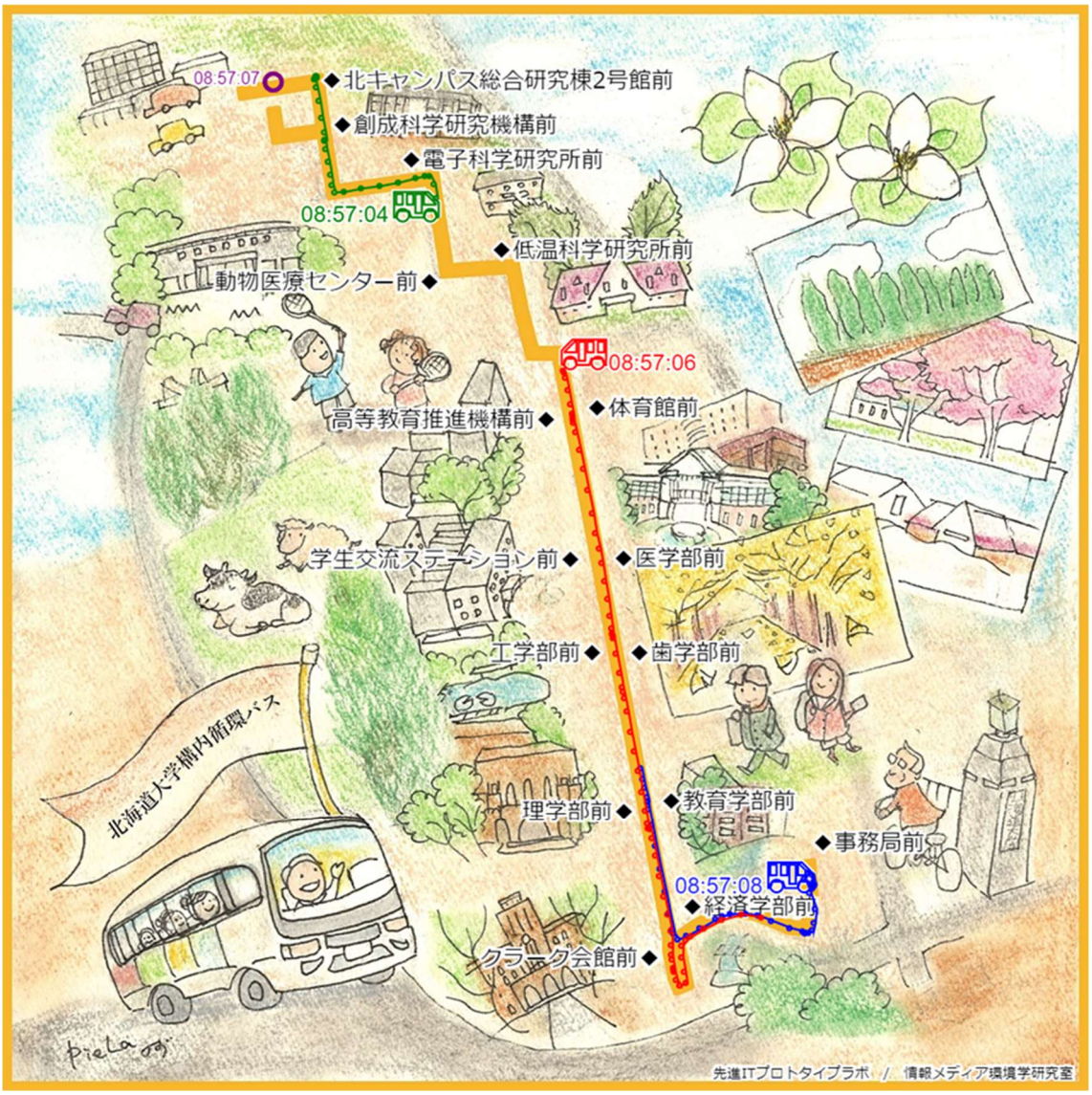

図５. MQTTRPC により開発された北大構内循環バスのロケーションサービス

5. むすび

本研究では IoT サービスのための RPC プロトコルとして MQTT を下層に用いる MQTTRPC プロトコルを提案し実装した。実装は Python3.0 と Javascript をターゲット環境として行ったので、エッジノードとブラウザを連携させたプログラム開発が可能になった。また、MQTTRPC ではクライアントプログラムがエッジノードを直接アクセスすることができるため、サーバーが存在しない、P2P 的なサービスとしてアプリケーションが開発できる。半導体 LSI の高速化、大容量化が進むことが予想されるため、本アプローチはコンパクトで拡張性の高いアプリケーション開発可能になる際に貢献できるものと考える。