Vol.2022-ARC-248 No.37
Vol.2022-SLDM-198 No.37
Vol.2022-EMB-59 No.37
2022/3/10

mROS 2：組込みデバイス向けの ROS 2ノード軽量実行環境

高瀬 英希[1,2,a)]　祐源 英俊[3]

概要：分散型のロボットシステムにおけるエッジデバイスの応答性やリアルタイム性の向上および消費電力の削減などへの期待から，ROS 2 への組込み技術の導入に注目が集まっている．本研究の目的は，組込みデバイス向けの高効率な ROS 2 通信方式およびメモリ軽量な実行環境を確立することである．提案する実行環境である mROS 2 は，主に組込み向けの軽量プロトコルスタックおよびリアルタイム OS から構成される．最大の利点は，ホストデバイス上の ROS 2 ノードとの通信において，既存の環境では必須であった仲介の役割を担う Agent ノードを不要にできることである．本研究では，目的の実現に求められる設計要件を整理し，効率的な通信処理を実現するためのソフトウェア構成および動作フローを設計する．また，mROS 2 通信ライブラリとして提供する API は，汎用 OS 向けの ROS 2 のクライアントライブラリと互換性を保つようにする．提案手法を STM32 NUCLEO-F767ZI ボード上に実装し，本研究成果が通信性能に優れた分散ロボットシステムの実現に貢献することを示す．

mROS 2: A Lightweight Runtime Environment of ROS 2 nodes for Embedded Devices

1. はじめに

ロボットシステムの開発を加速するプラットフォームである ROS（Robot Operating System）[1] の普及が進んでいる．ROS の主要な側面のひとつとして，出版購読型（publish/subscribe）の通信機能が提供されている．本機能によって，システムの機能単位であるノードの所望の組合せによって通信情報であるメッセージを交換しあう疎結合な分散システムを構築することができる．現在盛んに開発が進んでいる第 2 世代の ROS 2 では，DDS（Data Distribution Service）[2] が通信ミドルウェアとして採用されている [3]．この通信プロトコルである RTPS（Real-Time Publish Subscribe Protocol）[4][*1]では，通信相手の探索および通信経路の確立を自律的に行う機能を備えている．このノードおよび通信方式の自律性の高さから，ROS 2 および DDS の利点を IoT システム構築に応用することの機運が高まりつつある [5]．

ROS の本質は通信であり，これを採用するロボットシステムにおいては通信性能が重要となる．本研究では，特に通信性能として，応答性およびリアルタイム性に着目する．通信の応答性とは通信処理の遅延時間であり，小さいほど応答性が高いことを意味する．定められた時刻であるデッドラインまでに処理を完了できる能力であるリアルタイム性については，通信処理についてはその遅延時間の変動が小さいほど高いことを意味する．ROS 2 の通信機能を利用するロボットシステムは，一般には汎用 OS による実行環境が採用される．汎用 OS はシステム全体のスループット向上を主な目的として設計されているため，通信処理の応答性およびリアルタイム性を確保することが難しい．また，汎用 OS の稼働には高性能なデバイスを利用する必要があるため，システムコストおよび消費電力が大きくなる．

ロボットシステムの応答性やリアルタイム性の向上および消費電力の削減などの要求から，ROS 2 への組込み技術の導入に注目が集まっている．組込み向けのリアルタイム OS は，必要とする計算資源が限定的であり，優先度

[1] 東京大学
The University of Tokyo
[2] JST さきがけ
JST PRESTO
[3] 京都大学
Kyoto University
[a)] takasehideki@hal.ipc.i.u-tokyo.ac.jp

[*1] OMG（Object Management Group）による正式な規格名称は DDSI-RTPS（DDS Interoperability Wire Protocol）である．

ベースのスケジューリングによるリアルタイム性の確保を実現することに向いている．したがって，システム機能のエッジ部分に組込みデバイスを活用することで，上述の課題の解決が期待できる．既存の組込みデバイス向けの ROS 2 ノードの実行環境には，micro-ROS[6] がある．micro-ROS は RTPS の軽量規格である DDS-XRCE（DDS For Extremely Resource Constrained Environments）[7] の実装である Micro XRCE-DDS[8] を用いている．ホストデバイス上の ROS 2 ノードとの通信には，仲介の役割を担う Agent ノードの稼働が必要となる．これによって，ホストとエッジデバイス間の通信処理時間が増大し，応答性およびリアルタイム性が低下するおそれがある．

本研究の目的は，組込みデバイス向けの高効率な ROS 2 通信方式およびメモリ軽量な実行環境を確立することである．中規模の組込みデバイス上で実行されるプログラムが，汎用デバイス上の ROS 2 ノードと自律的に通信できるようにする．これによって，通信性能に優れた分散ロボットシステムの構築に貢献できることを目指す．

本稿では，組込みリアルタイムシステム向けの ROS 2 ノードの軽量実行環境である mROS 2 を提案する．本研究では，目的の実現に求められる設計要件を整理し，自律的かつ効率的な通信処理を実現するためのソフトウェア構成および動作フローを設計する．mROS 2 の構成には，RTPS の軽量実装である embeddedRTPS[9] およびリアルタイム OS である TOPPERS/ASP3[10] カーネルを採用する．これらを利用することで，通信時の応答性およびリアルタイム性が確保できるようにする．また，mROS 2 通信ライブラリとして提供する API は，汎用 OS 向けの ROS 2 の C++クライアントライブラリである rclcpp[11] と互換性を保つように設計する．これにより，ロボット開発における組込みデバイス採用時の生産性向上が見込める．

本稿の構成は，次のとおりである．まず 2 章では，準備として ROS の概要および ROS 2 の通信機能を解説する．次に 3 章では，本研究の方針を整理したのち，提案手法である mROS 2 について詳説する．4 章では提案手法を STM32 NUCLEO-F767ZI ボード上に実装し，既存の組込み向け実行環境との比較評価を実施する．最後に 5 章で本稿のまとめと今後の展望を示す．

2. 準備

2.1 ROS（Robot Operating System）

ROS[1] は，スタンフォード人工知能研究所の研究プロジェクトから移管された Willow Garage 社によって開発が始まったロボットソフトウェアの開発プラットフォームである．同社の PR2 というパーソナルロボットの開発環境として 2007 年に登場した．最初の正式なディストリビューション版である Box Turtle がリリースされたのは 2010 年 3 月である．現在では，Open Robotics が中心となって ROS の開発とメンテナンスが行われている．

ROS の利点は，分散システムの実現に向いた通信ミドルウェア，プロジェクト管理やデバッグおよびシミュレーションなどのためのツール群，再利用性の高い豊富な OSS のパッケージやライブラリ，および，世界規模の活発かつ継続的なオープンソースの開発コミュニティという 4 つの側面に集約される [12]．すでに 10 年以上の歴史があり，ロボット工学の発展に多大な貢献をしてきた．当初の主なユースケースは研究開発向けであったが，ソニー社のペットロボットである aibo[13] の事例に代表されるように，企業での商用製品に採用されることも多くなってきている．

ロボット開発を取り巻く背景や ROS のユースケースの変遷を受けて，2014 年より第 2 世代のバージョンである ROS 2 の開発が始まった [14]．ROS 1 の思想を踏襲しつつ，API 互換性も捨ててその基本設計から見直し，フルスクラッチから新たに実装し直されている．2015 年 8 月に最初の distribution がリリースされ，OSS コミュニティによる開発が活発に進んでいる．

2.2 ROS 2 の通信機能

ROS における基本的なノード間のデータ通信として，出版購読型の非同期な通信方式である `topic` が提供されている[*2]．本方式では，データの送信側を出版者（Publisher），受信側を購読者（Subscriber）とそれぞれ呼び，通信経路の定義であるトピックを介した通信が行われる．トピックで送受信されるデータをメッセージと呼び，例えば両車輪の角速度と回転量や現在位置の 3 次元座標など，基本型を組合せた任意の型を定義または利用することができる．同じ名前のトピックに対して，任意の個数や種類のノードが任意のタイミングで登録や変更，削除ができる．さらにメッセージの型が同一であるときにデータ通信が行われる．出版側は任意のタイミングで動作でき，非同期に動作する購読側では対応するコールバック関数が実行される．ノード同士の結合が疎になるため，システム機能の追加や削除が容易であり，システム構成を柔軟に変更できる．

ROS 2 の通信ミドルウェアである DDS[2] およびこの通信プロトコルである RTPS[4] では，通信相手の探索および通信経路の確立が自律的に行われる．これを実現するため，RTPS には，SPDP（Simple Participant Discover Protocol）および SEDP（Simple Endpoint Discover Protocol）が備えられている．ここで，RTPS では，ROS 2 のノードに相当するものを Participant と呼ぶ．通信相手を探索するために自身の情報を送信するモジュールを Writer，他の Participant から情報を受信するモジュールを Reader とそれぞれ呼ぶ．Endpoint は，Participant の出版および購読の通

[*2] 1 対 1 の同期式の RPC 通信である `service` やフィードバック付き RPC 通信である `action`，多変量辞書である `parameter` も用意されている．

Vol.2022-ARC-248 No.37
Vol.2022-SLDM-198 No.37
Vol.2022-EMB-59 No.37
2022/3/10

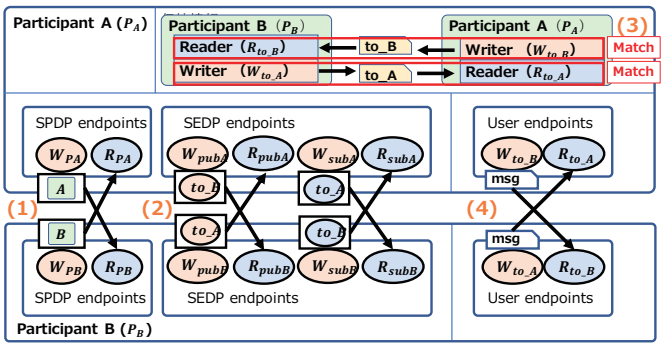

図 1 RTPS における通信相手の探索および通信経路の流れ

信端点となるモジュールである．**図 1** は，`Participant A` および `Participant B` が互いに通信相手の探索と通信経路の確立を行う例であり，次に示す手順が行われる．

(1) Participant の Writer から，通信相手を探索する SPDP によって，自身の Participant 情報を他の Readers に対してマルチキャストで送信する．この送信は，その生成時およびそれ以降に定期的に行われる．
(2) 情報を受信した Participant は，通信経路を探索する SEDP によって，自身の出版および購読に関わる Endpoints 情報を，その Writer から送信元の Reader にユニキャストで返送する．
(3) 返送されてきた Endpoints 情報と自身の情報を照合し，Publisher と Subscriber の対として Endpoints 情報が合致した相手同士で，通信経路が確立される．
(4) 以降から，それらの Participants 間で，メッセージの出版購読通信が可能となる．

RTPS は OSI 参照モデルにおける transport 層に位置し，UDP/IP 上に実装される．UDP 通信にはパケット到着などについて不確実性があるが，ROS 2 および DDS ではこれを補助する QoS（Quality of Service）制御の機能がある．

ROS 2 で DDS が採用された理由として，保守対象のソースコードを削減できることや，厳格かつ明確な仕様に依存できることが挙げられる．各ベンダからその仕様に基づいた複数の実装が提供されており，それらは高い互換性がある．ROS 2 のソフトウェア構造は階層化されており，実装機能やライセンス形態に合わせて任意の DDS を選択できる．本稿執筆時点での最新の長期サポート版である ROS 2 Foxy では，eProsima Fast-RTPS，Eclipse Cyclone DDS および RTI Connext が公式な DDS として対応している [15]．

3. mROS 2

本研究の目的は，組込みデバイス向けの高効率な ROS 2 通信方式およびメモリ軽量な実行環境を確立することである．ROS 2 を採用するロボットシステムに組込み技術を導入することにより，分散システムのエッジデバイスにおける応答性やリアルタイム性の向上および消費電力の削減が期待できる．本章では，目的を達成するための本研究の方針を議論したのち，提案手法である mROS 2 について詳説する．設計要件を整理した上で，ホストデバイスと組込みデバイスの間で自律的かつ効率的な ROS 2 通信処理を実現するソフトウェア構成および動作フローの設計を示す．

3.1 方針

まず，組込みデバイス上で動作するアプリケーションは，汎用 OS 上で実行される ROS 2 のノードと自律的に通信できることが望ましい．本研究における通信の自律性とは，2.2 節で示したとおり，出版購読型通信の相手および経路を自律的に探索できることを意味する．この ROS 2 および RTPS の利点が，組込み技術の導入時にも保持されるようにする．1 章で触れたとおり，既存の組込みデバイス向けの ROS 2 ノードの実行環境である micro-ROS[6] および Micro XRCE-DDS[8] では，ホストデバイス上の ROS 2 ノードとの通信に Agent ノードの稼働が必要となる．通信の仲介の役割を担うため，Agent ノードでは，ホスト上の ROS 2 ノードとは RTPS に則った通信，組込みデバイス上のノードとは XRCE [7] に則った通信がそれぞれ行われる．これらの通信処理および中継に伴う処理遅延が発生するため，応答性およびリアルタイム性の低下に繋がりうる．また，複数の組込みデバイスを用いる場合には，Agent ノードは分散システム全体にとっての単一障害点となる．

次に，効率的な通信処理を実現するため，中規模程度の性能を持つ組込みデバイスを採用することとする．中規模程度とは，ネットワークインタフェースを有し，組込み向けのリアルタイム OS および通信プロトコルスタックが動作可能であることを意味する．リアルタイム OS の採用によって，省資源のメモリ量で高いリアルタイム性を確保できることが見込まれる．また，本研究の目的のためには，ネットワークインタフェースおよび通信プロトコルスタックは本質的に必須である．小規模な低性能の組込みデバイスではこれらを満足することができない．処理に関する高い応答性は期待できるものの，リアルタイム OS による支援が見込めないため，相応のプログラミング技術が必要となる．また，大規模な高性能の組込みデバイスでは，部品コストおよび消費電力が大きくなるため，組込み技術を導入する本来の意義を得ることができなくなる．

開発生産性の観点から，組込みデバイス上のアプリケーション実装に関するプログラミングモデルも考慮するべきである．ROS 2 の標準的な API との互換性を提供できれば，既存の ROS 2 のプログラム資産をそのまま組込みデバイス上のアプリケーションとして容易に流用することができるようになる．また，システム開発者に求められる学習コストも抑えられることが期待される．

本稿では，非同期式の出版購読型である `topic` 通信方式のみを対象とし，メッセージの型については基本型（build-in-type）のみを扱うこととする．これらは実用的には厳し

Vol.2022-ARC-248 No.37
Vol.2022-SLDM-198 No.37
Vol.2022-EMB-59 No.37
2022/3/10

い制約であるが，本研究の成果が基礎技術として形成できれば，任意型のメッセージへの対応ならびに通信方式の機能拡張も可能になるものと考えられる．

3.2 設計要件

3.1 節の議論を踏まえ，本研究で提案する軽量実行環境である mROS 2 に求められる設計要件を整理する．

通信の自律性を確保するため，RTPS の仕様に則った通信が実現できるようにする．すなわち，Micro XRCE-DDS における Agent ノードのような仲介の仕組みは不要となるようにする．ただし，本研究の対象は中規模の組込みデバイスであり，Linux に代表される汎用 OS の稼働は想定しない．このため，汎用 OS が提供する機能やシステムコールに依存しないプロトコルスタックを採用する必要がある．

通信性能を向上させるため，品質の高いリアルタイム OS を採用する．特にリアルタイム性を確保するため，優先度ベースの適切なタスクスケジューラを持つことが求められる．また，メモリ軽量性を実現するためには，採用する通信プロトコルスタックおよびリアルタイム OS は，C または C++による実装のものであることが望ましい．

対象とする組込みデバイスの性能および規模としては，通信プロトコルスタックおよびリアルタイム OS を実行できる処理能力を有するプロセッサ，および，これらを搭載できるメモリ容量が必要となる．より具体的には，最大動作周波数が数百 MHz 程度，ROM 容量および RAM 容量がそれぞれ数 MB 程度と定める．ネットワーク通信のためのインタフェースとしては Ethernet を想定し，RJ45 コネクタを有するデバイスを対象とする．

プログラミングモデルについては，ROS 2 における C++のクライアントライブラリである rclcpp[11] と互換性のある，基本的な出版購読型の通信方式に関する API が提供できることを要件とする．つまり，mROS 2 の通信ライブラリとして C++による API を提供できるようにする．ただし，rclcpp では汎用 OS 上での動作を想定した現代的な機能を活用した豊富な機能が提供されているため，本研究では，これらのうち，計算資源が限定的な組込みデバイスで動作可能であるものについてのみ対応することとする．

3.3 ソフトウェア構成

図 2 に，mROS 2 のソフトウェア構成を示す．ユーザアプリケーションからの階層順で，mROS 2 通信ライブラリ，通信プロトコルスタック，リアルタイム OS，および，ハードウェア抽象化ライブラリによって構成される．

mROS 2 通信ライブラリは，ユーザアプリケーションに対して ROS 2 通信の `topic` に関する基本的な API を提供する．**表 1** に，mROS 2 通信ライブラリが提供する主な API を示す．rclcpp の豊富な機能から最小限のものを選定しているが，一定の互換性を保つように設計しており，組

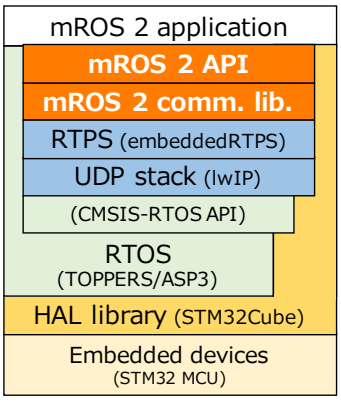

図 2 mROS 2 のソフトウェア構成

込み技術を導入するシステム開発者は，ROS 2 のプログラミングスタイルを参考にしながら，C++によって mROS 2 のアプリケーションを実装できる．

通信プロトコルスタックには，C++で実装された embeddedRTPS[9] を採用する．これを採用する理由として，RTPS における SPDP および SEDP が実装されており，通信の自律性を確保できることが挙げられる．また，メモリ領域の静的確保など，計算資源の限定的な組込みデバイス上での稼働を想定した設計がなされている点．ROS 2 で代表的な Fast-RTPS と疎通確認されており親和性が高い点，OSS として継続的に開発されている点なども，本研究の要件に適している．UDP については組込み向けの C による軽量実装である lwIP[16] を，RTPS メッセージのシリアライズには eProsima Micro CDR[17] をそれぞれ用いている．

リアルタイム OS には，TOPPERS/ASP3 カーネル [10] を採用する．本カーネルでは，高分解能タイマやティックレスの低消費電力な処理遅延機能など，高いリアルタイム性および安全性が求められる軽量な組込みシステムに適した設計がなされている．プログラムサイズが 1MB 程度までのシステムを適用対象と想定しており，mROS 2 の設計要件にも合致する．ただし，embeddedRTPS および内包される lwIP は CMSIS-RTOS API[18] に依存しているため，それぞれの API 差分を吸収するラッパレイヤを用意した．

表 1 mROS 2 通信ライブラリで提供される主な API
（名前空間 `mros2::`は除外している．）

return type	name	args
`void`	`init`	`argc` `*argv[]`
`Node`	`Node::create_node`	`node_name`
`Publisher`	`Node::create_publisher`	`topic_name` `qos_keep_last`
`void`	`Publisher::publish`	`&msg`

`Subscriber`	`Node::create_subscription`	`topic_name`
		`qos_keep_last`
		`*fp_callback`

Vol.2022-ARC-248 No.37
Vol.2022-SLDM-198 No.37
Vol.2022-EMB-59 No.37
2022/3/10

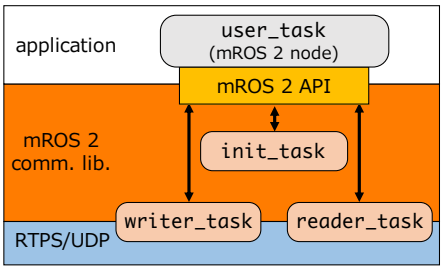

図 3 mROS 2 の通信機能を提供するタスク構成

ハードウェア抽象化ライブラリには，STM32 マイコン向けの STM32Cube ライブラリ [19] を用いる．すなわち mROS 2 は，実行環境として STMicroelectronics 社のマイコンのみをサポートすることとなる．ただし，同社のマイコンはすでに幅広い製品で利用されており，本ライブラリはマイコンファミリ間の互換性が高いため，この制限は mROS 2 の採用が期待される分野を制限するものではないことに注意されたい．なお，ソフトウェア構成の下位層にあるリアルタイム OS および HAL ライブラリについては，他のモジュールに変更することは可能であると考えられる．また，アプリケーションからは ASP3 カーネルおよび STM32Cube ライブラリの API を直接利用できるため，ROS 2 のノード実装に組込みデバイス向けのプログラミングを実現することができる．

3.4 動作フロー

本節では，メッセージの効率的な送受信処理を実現するための mROS 2 のタスク構成および動作フローを示す．

3.4.1 タスク構成

図 3 に示すとおり，mROS 2 の通信機能は，`init_task`，`writer_task` および `reader_task` のタスク群によって構成される．`user_task` は，システム開発者が実装するタスクである．なお，mROS 2 実行環境には，ネットワークインタフェースの管理のためのタスクやカーネル内のシステムタスクも存在するが，本稿ではこれらの説明は省略する．

`user_task` は，ROS 2 におけるノードに相当する，出版購読型の通信を行う mROS 2 のノード・アプリケーションである．mROS 2 の API を呼び出すことで，ノードとしての登録や出版および購読の処理を行うことができる．図 3 は 1 つのタスクで出版および購読の双方を行う構成の図示であるが，TOPPERS/ASP3 カーネルのプログラミングモデルに従って，mROS 2 ノードの機能を複数のタスクとして実装することもできる．

`init_task` は，ROS 2 としてのノード情報の初期化を行う．具体的には，`mros2::init()` が呼ばれたときに，対象の組込みデバイスを RTPS の Participant として登録する．初期化処理を行ったのち，本タスクは休止状態に移行する．なお，mROS 2 では，1 つのデバイスに対して単一の RTPS Participant として動作することを想定している．また，

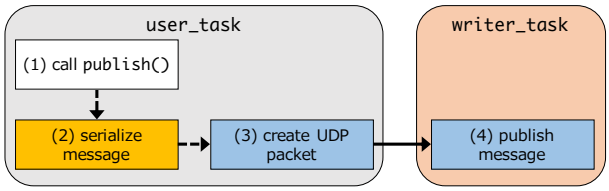

図 4 mROS 2 におけるメッセージの出版処理

embeddedRTPS では，単一の Domain クラスから Participant，Writer および Reader の機能に相当するインスタンスが生成される．すなわち，本初期化処理によって，embeddedRTPS の提供する出版購読型の通信機能が利用できるようになる．これらの機能は，`mros2::Node::create_node()` の呼び出しによって `user_task` に紐付けられる．

`writer_task` および `reader_task` は，出版および購読に関する処理を担う．これらのタスクは，`user_task` からの mROS 2 API を介した依頼を受け，embeddedRTPS の該当する機能を実行する．`writer_task` は，出版依頼を受けて起動し，RTPS メッセージのデータを UDP パケットとして送信する．`reader_task` は，UDP パケットの受信時に起動し，RTPS のパケットであればこれを順に処理する．

3.4.2 メッセージの出版処理

まず，`mros2::Node::create_publisher()` によって mROS 2 ノードを出版者として登録し，mROS 2 の実行環境において出版処理を行えるようにする．本 API は関数テンプレートとして定義されており，テンプレート仮引数にはメッセージの型，引数にはトピック名および RTPS における QoS 制御の履歴長を指定する．これによって，`user_task` は紐付けられた embeddedRTPS の出版機能を利用できるようになる．

図 4 に示す出版時の動作フローは，次のとおりである．

(1) `user_task` から `mros2::Publisher.publish()` を呼び出す．引数にはメッセージ情報が格納されたオブジェクトのポインタを渡す．
(2) mROS 2 通信ライブラリ内でメッセージのシリアライズを行い，RTPS に則った形式に変換する．
(3) embeddedRTPS の提供する機能によって UDP パケットを作成し，タスク間通信で `writer_task` に出版処理を依頼する．
(4) `writer_task` が実行され，出版処理を行う．

mROS 2 通信ライブラリにおける通信処理の効率化のために `writer_task` を設計した意図について説明する．ネットワーク通信は I/O バウンドな処理であり，プロセッサにおけるアプリケーションの実行と並行に行うことができる．このため，embeddedRTPS および lwIP におけるメッセージおよび UDP パケットの送信処理に関しては，`user_task` から分離して `writer_task` で実行されるようにしている．

3.4.3 メッセージの購読処理

`mros2::Node::create_subscription()` によってノー

Vol.2022-ARC-248 No.37
Vol.2022-SLDM-198 No.37
Vol.2022-EMB-59 No.37
2022/3/10

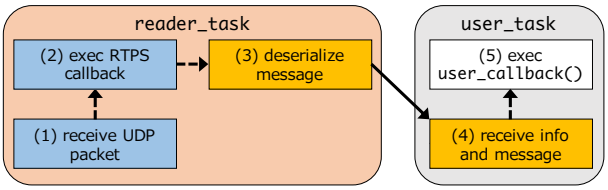

図 5 mROS 2 におけるメッセージの購読処理

ドを購読者として登録し，メッセージの購読処理を行えるようにする．本 API も同様に関数テンプレートとして定義されており，テンプレート仮引数にはメッセージの型，引数にはトピック名および RTPS における QoS 制御の履歴長に加えて，購読時に実行されるコールバック関数を指定する．これによって，user_task は紐付けられた embeddedRTPS の購読機能を利用できるようになる．

図 5 に示す購読時の動作フローは，次のとおりである．

(1) reader_task が UDP パケットを受信する．
(2) init_task 時に Participant 情報として登録されていた embeddedRTPS 内のコールバック関数が実行され，RTPS に則った形式に変換する．
(3) mROS 2 通信ライブラリ内で RTPS パケットのデシリアライズを行い，メッセージに変換する．
(4) user_task に対して，タスク間通信によって登録されていたコールバック関数および購読されたメッセージのオブジェクトのポインタを渡す．
(5) user_task のコンテキストで，メッセージのオブジェクトのポインタを引数としてコールバック関数を実行する．

reader_task の設計意図について説明する．DDS におけるメッセージの出版購読型の通信は非同期に行われるため，user_task で購読を待ち受けるには CPU 使用権を専有する必要がある．また，メッセージの購読時には対応するコールバック関数を実行する必要があり，この処理時間が長くなる場合には次のメッセージの購読を待ち合わせることができなくなる．このため，UDP パケットの到着を高い優先度で定期的に監視する reader_task を用意し，到着時のみにメッセージの購読処理が行えるようにしている．

3.5 mROS 2 ノードの実装例

図 6 に，mROS 2 通信ライブラリの提供する API を用いたノードの実装例を示す．この例は，いわばエコーリプライの処理を行うノードであり，1 つの組込みデバイスおよびノードが出版と購読の双方の通信を行う．to_stm のトピックから文字列型のメッセージを購読し，これをそのままコールバック関数内で to_linux のトピックに対して出版する．

4. 評価

本章では，提案手法の有効性を，通信性能およびプログラムサイズの観点から定量的に評価する．提案する軽量実行環境 mROS 2 は，STM32 NUCLEO-F767ZI[20] を対象として実装した．ホスト開発環境は，Ubuntu 18.04, STM32CubeIDE 1.5.0，および，arm-none-eabi-gcc 7.3.1 である．embeddedRTPS は commit hash a52ed04，TOPPERS/ASP3 カーネルは Release 3.5.0，HAL ライブラリは STM32Cube FW_F7 v1.16.0 を用いた．本稿における評価対象は mros2-asp3-f767zi v0.1-rc[*3]である．

```cpp
#include "app.h"
#include "mros2.h"
#include "std_msgs/msg/string.hpp"

#include "stm32f7xx_nucleo_144.h"

mros2::Publisher pub;
mros2::Subscriber sub;

void userCallback(std_msgs::msg::String *msg)
{
  MROS2_INFO("subscribed msg: '%s'", msg->data.c_str());
  MROS2_INFO("publishing msg: '%s'", msg->data.c_str());
  pub.publish(*msg);
}

int main(int argc, char * argv[])
{
  MROS2_INFO("mROS 2 application is started");

  mros2::init(argc, argv);
  MROS2_DEBUG("mROS 2 initialization is completed");
  BSP_LED_Toggle(LED1);

  mros2::Node node = mros2::Node::create_node("mros2_node");
  pub = node.create_publisher<std_msgs::msg::String>("to_linux", 10);
  sub = node.create_subscription("to_stm", 10, userCallback);

  MROS2_INFO("ready to pub/sub message");
  mros2::spin();
  BSP_LED_Toggle(LED3);
}

void main_task(void)
{
  main(0, NULL);
}
```

図 6 mROS 2 ノードの実装例

4.1 評価環境

表 2 に評価環境を示す．既存の組込みデバイス向けの実行環境に対する優位性を議論するため，micro-ROS[6] に対する定量的評価も実施した．micro-ROS のリアルタイム OS には NuttX を用いた．mROS 2 ノードと通信する汎用デバイスには，Raspberry Pi 3 Model B を用いた．ROS 2 のディストリビューションは，Ubuntu 18.04 で動作する Dashing Diademata である．micro-ROS における Agent

表 2 評価環境

embedded device	STM32 NUCLEO-F767ZI
processor	Arm Cortex-M7 216 MHz
memory	512 KB RAM / 2 MB Flash
general-purpose device	Raspberry Pi Model B
processor	Arm Cortex-A53 4 core 1.2 GHz
memory	8 GB RAM

[*3] https://github.com/mROS-base/mros2-asp3-f767zi/tree/v0.1-rc

Vol.2022-ARC-248 No.37
Vol.2022-SLDM-198 No.37
Vol.2022-EMB-59 No.37
2022/3/10

ノードには Micro-XRCE-Agent v1.3.0 を用いた．

4.2 通信性能

通信性能の評価には，2 つのデバイス間で 32 ビット整数値型 `std_msgs::msg::Int32` のメッセージをエコーバックするアプリケーションを用いた．ホストデバイス上のノードからメッセージを出版し，エッジノードではこれを購読してそのままのメッセージを出版し，ホストノードでこれを購読する．メッセージはホストデバイスから 1 秒間隔で計 200 回出版し，ホストデバイス上でエコーバックに掛かるラウンドトリップタイム（RTT）を計測した．時間計測には `std::chrono::high_resolution_clock` を用いた．

評価対象のシステム環境は，**図 7** に示す 4 種類のものを対象とした．各デバイスはイントラネットを構成したネットワークルータを介して有線接続した．環境 (A) は ROS 2 を用いた汎用デバイス同士の通信であり，残りは汎用デバイスをホストとして組込み向け実行環境を搭載したデバイスをエッジデバイスとした通信となる．環境 (B) および (C) はいずれも micro-ROS を用いたものであるが，Agent ノードの配置が異なることに注意されたい．micro-ROS の API は C のクライアントライブラリである rclc のみが利用可能であるため，評価アプリケーションは C で実装した．本研究で提案する mROS 2 を採用したシステム環境は (D) となる．本環境では，購読および出版の処理は，それぞれタスクを分けて実装した．

評価結果を箱ひげ図として**図 8** に示す．図 8 から，提案手法である環境 (D) の通信性能が明らかに優位であることがわかる．環境 (D) の RTT の最大値は 1323μs であり，他の環境での最小値を下回っている．汎用デバイス同士の通信である環境 (A) よりも優れた応答性が得られていることは特筆すべきである．micro-ROS の環境 (B) および (C) では，Agent ノードの配置によって通信性能に差異が見られた．提案手法では通信の仲介の仕組みが不要であるため，このようなことは発生しない．次に，通信処理のリアルタイム性について議論するため，RTT 値の変動に注目する．図 8 の四分位範囲をみると，やはり提案手法である環境 (D) の結果が優れている．環境 (B) では外れ値が観測されているが，このような傾向も生じていない．以上のことから，提案する mROS 2 は分散システムのエッジデバイスにおける通信性能の向上に貢献できることが示された．

4.3 プログラムサイズ

実行環境の軽量性を議論するため，前節の通信性能の評価に用いたアプリケーションを対象として評価する．環境 (B) および (C) の micro-ROS および提案手法である環境 (D) の mROS 2 のそれぞれについて，エッジノードとして実装したエコーバックアプリケーションバイナリを `size` コマンドで計測した．いずれの環境でも実行環境に加えて

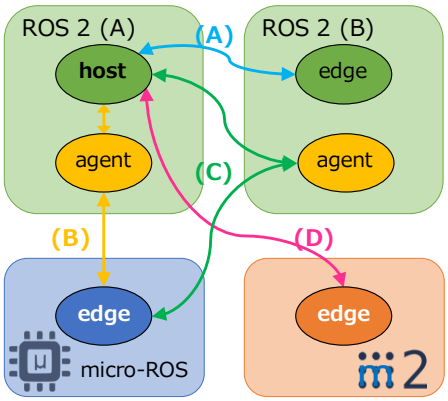

図 7 通信性能の評価に用いるシステム環境

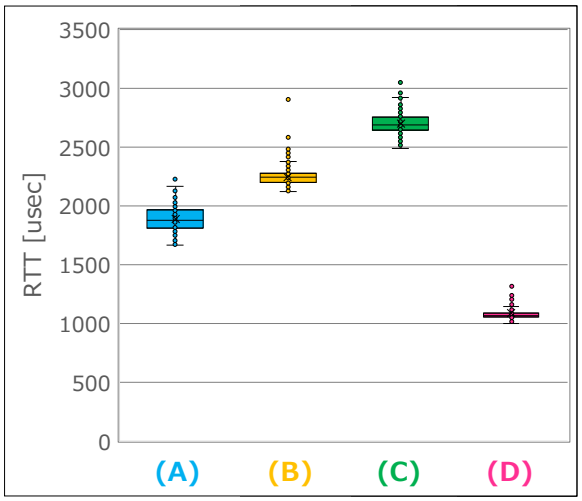

図 8 ホストデバイスにおける RTT の評価結果

アプリケーションのメモリ領域も評価値に含まれるが，本アプリケーションは単純かつ小規模なものであるため，大きな影響は無いものと考えている．

プログラムサイズの評価結果を**表 3** に示す．mROS 2 のプログラムサイズは，合計で約 210 KB となった．FLASH に配置される text 領域は約 88 KB，RAM に配置される data 領域および bss 領域の合計は約 121.4 KB である．いずれも 3.2 節で挙げた設計要件を満たしており，本研究の対象とした STM32 NUCLEO-F767ZI のメモリにも十分に収まる容量となった．

micro-ROS と比較すると data 領域は大幅に増加しているが，これは TOPPERS/ASP3 カーネルではカーネルオブジェクトなどをコンパイル時に静的に生成する方針を採っているためである．いっぽう，micro-ROS のリアルタイム OS として本研究で採用した NuttX では，ビルド時に指定したアプリケーションをバイナリに含め，実行時

表 3 バイナリサイズの評価 [Byte]

	text	data	bss	dec
micro-ROS	439,660	1,206	306,092	746,958
mROS 2	87,603	16,632	104,768	209,003

にシェルから呼び出す形式でアプリケーションが実行されるため，実行時にあらかじめ初期化される変数が少なくなる．micro-ROS の text 領域が大きい理由は，QoS 制御など ROS 2 との互換機能がすでに実装されていることが挙げられる．また，NuttX が POSIX 準拠のインタラクティブシェルを持つなど，比較的高機能なリアルタイム OS であることも影響している．本研究では，基本的かつ最小限な出版購読型の通信機能に限定して mROS 2 の設計を進めた．今後，本研究で達成したメモリ軽量性を損なわないように，機能拡充を進めていく予定である．

5. おわりに

本研究では，分散型のロボットシステムにおけるエッジデバイスの通信性能を向上させるため，組込みデバイス向けの軽量実行環境である mROS 2 を提案した．mROS 2 は，中規模の組込みデバイス上で実行されるプログラムに対して，汎用デバイス上の ROS 2 ノードと自律的な出版購読型の通信機能を提供する．mROS 2 のソフトウェア構成には，RTPS の軽量実装である embeddedRTPS およびリアルタイム OS である TOPPERS/ASP3 カーネルを主に採用した．通信ライブラリとしての効率的な処理を実現するために，ソフトウェア構成および動作フローを設計した．また，mROS 2 通信ライブラリとして提供する API は，汎用 OS 向けの rclcpp と互換性を保つように設計した．

STM32 NUCLEO-F767ZI を対象として提案手法を実装し，その有効性を定量的に評価したところ，高い通信性能およびメモリ軽量性が発揮できることが確認された．本研究の成果は，通信処理に関する応答性およびリアルタイム性に優れた分散ロボットシステムの実現に貢献できる．

今後の方針として，まず，mROS 2 の最新バージョンによって評価することが挙げられる．本稿では v0.1-rc の ROS 2 Dashing での評価結果を報告したが，これ以降には，ディレクトリ階層およびソースコードの大規模なリファクタリング，Mbod OS のサポートなどの更新およびリリースを実施している．さらに，v0.2.1 では ROS 2 の最新の長期サポート版である Foxy Fitzroy の対応も完了している．次に，任意型のメッセージへの対応ならびに通信方式の機能拡張が挙げられる．前者については，すでに研究成果を挙げつつあり，文献 [21] にて報告される．また，後者をはじめとした様々な機能拡張や品質向上に取り組んでいく予定である．

mROS 2 のソースコードは，GitHub にて OSS として公開している（`https://github.com/mROS-base/mros2`）．ご興味のある方は，ぜひ試用いただき，研究開発への参画や採用を検討いただけたら幸甚である．

謝辞 本研究の一部は，JST さきがけ JPMJPR18M8 の支援ならびに国立研究開発法人情報通信研究機構の委託研究（04001）により得られたものである．

参考文献

[1] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y.: ROS: an open-source Robot Operating System, *Proc. of ICRA workshop on open source software*, No. 3.2, pp. 1–6 (2009).
[2] Object Management Group: About the Data Distribution Service Specification Version 1.4 (online), `https://www.omg.org/spec/DDS/` (2022.01.29).
[3] Woodall, W.: ROS on DDS (online), `https://design.ros2.org/articles/ros_on_dds.html` (2022.01.29).
[4] Object Management Group: About the DDS Interoperability Wire Protocol Version 2.5 (online), `https://www.omg.org/spec/DDSI-RTPS/` (2022.01.29).
[5] 高瀬英希：ROS (Robot Operating System) の紹介と IoT/IOT 分野への展開，RICC-PIoT workshop 2022.
[6] micro-ROS | ROS 2 for microcontrollers (online), `https://micro.ros.org/` (2022.01.29).
[7] Object Management Group: About the DDS For Extremely Resource Constrained Environments Specification Version 1.0 (online), `https://www.omg.org/spec/DDS-XRCE/` (2022.01.29).
[8] eProsima Micro XRCE-DDS (online), `https://micro-xrce-dds.docs.eprosima.com` (2022.01.29).
[9] Kampmann, A., Wüstenberg, A., Alrifaee B. and Kowalewski, S: A Portable Implementation of the Real-Time Publish-Subscribe Protocol for Microcontrollers in Distributed Robotic Applications, *Proc. of 2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 443–448, (2019).
[10] TOPPERS/ASP3 kernel (online), `https://toppers.jp/asp3-kernel.html` (2022.01.29).
[11] rclcpp: ROS Client Library for C++ (online), `https://docs.ros2.org/latest/api/rclcpp/` (2022.01.29).
[12] 近藤豊：ROS2 ではじめよう 次世代ロボットプログラミング，技術評論社 (2019).
[13] 水上智雄，藤田智哉：aibo における ROS の利用とソニーの取り組み，ROSCon JP 2018 (2018).
[14] Gerkey, B.: Why ROS 2? (online), `https://design.ros2.org/articles/why_ros2.html` (2022.01.29).
[15] REP 2000 – ROS 2 Releases and Target Platforms (online), `https://www.ros.org/reps/rep-2000.html` (2022.01.29).
[16] Umang, M.: lwIP: Light Weight IP, The Goerge Washington University (2011).
[17] eProsima Micro CDR (online), `https://github.com/eProsima/Micro-CDR` (2022.01.29).
[18] Arm Ltd.: Common Microcontroller Software Interface Standard (CMSIS) — Arm Developer (online), `https://developer.arm.com/tools-and-software/embedded/cmsis` (2022.01.29).
[19] STMicroelectronics: STM32Cube Development Software (online), `https://www.st.com/ja/ecosystems/stm32cube.html` (2022.01.29).
[20] STMicroelectronics: NUCLEO-F767ZI (online), `https://www.st.com/en/evaluation-tools/nucleo-f767zi.html` (2022.01.29).
[21] 檜原 陽一郎，中村 宏，高瀬 英希：ROS 2 ノード軽量実行環境 mROS 2 における任意型メッセージの通信処理方式，情報処理学会研究報告，発表予定 (2022).