Vol.2020-EMB-54 No.5
2020/6/26

リアルタイムOSによるROSノード軽量実行環境の定量的評価

祐源 英俊[1,a)]　高瀬 英希[1,2,b)]

概要：近年, ロボットソフトウェアの開発を支援するプラットフォームである ROS が関心を集めている. ROS を活用すると, 豊富な既存プログラム資源を活用することができ, また分散システムの制御ソフトウェアを効率的に構築することができる. この ROS を利用するシステムにおいて, 組込み機器を活用する手法が特に注目されている. 組込み機器の活用により, アプリケーションのリアルタイム性性能確保などの恩恵が受けられると期待される. そこで本研究では, リアルタイム OS 上で実行され, 組込み機器に搭載可能な, ROS ノードの軽量実行環境に対し, 定量的な性能評価を行う. 評価は, ネットワーク通信を行う mROS, および, シリアル通信を行う rosserial の 2 手法を対象として行う.

Quantitative evaluation of ROS node lightweight execution environment by real-time OS

1. はじめに

近年, ロボットソフトウェアの開発を支援するソフトウェアプラットフォームである ROS が注目を集めている. ROS の提供する機能の一つとして, システムを構成するソフトウェアコンポーネント同士の通信を可能にする, 通信ミドルウェアの機能がある. この機能を活用することにより, 本来は複雑かつ広範な知識が要求されるロボットシステムを, ノードと呼ばれる機能部品プログラムの組み合わせによって容易に表現し, 構築することができる. ROS の提供するデータ通信の機能は, 出版購読方式に基づいている. このため, 通信を行うノード同士の結合は疎であり, 複数の機器がネットワーク通信により協調動作する分散システムの構築に適している. さらに, ROS はオープンソースで開発されており, 利用可能な既存のプログラム資源が豊富である. このように配布されているプログラムを利用することにより, 複雑なシステムであっても非常に効率的に開発することができる.

ROS による通信機能は, Linux のミドルウェアとして提供される. このため, ROS を活用して構築したシステムを実行するためには, Linux 環境が必要である. Linux 環境は, メモリサイズが大きくなり, 軽量なリアルタイムカーネルと比べ, リアルタイム性能が乏しい. なおここでいうリアルタイム性能とは, タスクを決められた時間内に完了させることができる能力のことを指す. さらに, 消費電力, 計算能力, およびメモリサイズなど, 要求される計算資源も高い. このため, ROS を活用するためには, 計算資源が豊富で, 消費電力が高いデバイスを採用せざるを得ない.

これを受けて近年, ROS を用いるシステムにおいて, 組込み機器を活用する手法が注目を集めている. ROS を活用した分散システムの一部ノードを, 計算資源および消費電力が小さい組込み機器において実行する. これにより, リアルタイム性能, 消費電力, およびコストなどの削減を行うことができる. また, ROS のインターフェースに則った開発が行えることにより, ROS を用いる開発者が, 低いコストで組込み機器を利用するシステムを開発することができるという利点もある.

そこで本研究では, 組込みシステム上で ROS のノードを実行できる実行環境に対し, 定量的な性能評価を行う. 評価は, ネットワーク通信を行う mROS, および, シリアル通信を行う rosserial を対象とする. それぞれについて, 通信速度および ROS ノードプログラム移植時のコストを, 定量的に評価し, 比較する. 本研究による定量的な評価結果が,

[1] 京都大学
[2] JST さきがけ
[a)] emb@lab3.kuis.kyoto-u.ac.jp
[b)] takase@i.kyoto-u.ac.jp

組込みシステム開発現場への ROS 導入のきっかけとなり, 組込みシステムの開発効率向上, および, ROS を活用するシステムへの組込み機器の導入が促進されることが期待される.

本文献の構成は以下の通りである. 2 章で, 背景知識として ROS, mROS, および rosserial について説明する. 3 章で, 本研究で行う評価についての説明, および, 評価結果を掲載する. 4 章で, 評価結果について考察し議論する. 5 章で, 本文献のまとめおよび今後の展望を述べる.

2. ROS への組込み機器の導入

本章では, ROS について説明し, ROS への組込み機器技術の導入について, その利点および問題点について述べる.

2.1 ROS

ROS は, ロボットシステムの開発を支援するソフトウェアプラットフォームである. 元々研究においてプロトタイプ機を効率よく開発するため, スタンフォード大学および Willow Garage 社によって開発された. 現在は, OSRF（Open Source Robotics Foundation）によって, 維持管理, リリース, および産業界への導入促進が行われている.

ROS は, ロボットシステム開発のための通信ミドルウェア, ビルドツール, デバッグツール, および, シミュレーションツールなどを提供する. 中でも代表的な機能として, 出版購読モデルに基づいた通信機能をシステムのコンポーネントに提供するものが挙げられる. これは, トピックと呼ばれる通信チャネルを介し, データを送信する出版者, および, データを受信する購読者とが通信を行うものである. ROS において, この通信で送受信される通信情報はメッセージと呼ばれる.

この, 出版購読モデルに基づいた通信は, 全てトピックを介して行われる. このため, 通信動作を行うさらにこの仕様のため, システム内のノード同士の結合が弱くなっている. したがって, システムに対するノードの追加および削除を非常に容易に行え, システムの構成を柔軟に変更することができる.

一方, この ROS の通信機能は, Linux の通信ミドルウェアとして提供されている. このため, ROS のノードを実行するためには原則, Linux 環境が必須である.

なお, 現在は ROS のほか, 次世代版 ROS である ROS2 もリリースされている. ROS2 は通信動作の実装を一新し, 組込み機器など比較的リソースの少ないデバイスでも実行することを想定したものとなっている. しかし, 利用者の数および利用可能なプログラム資産は, 依然として ROS2 より ROS の方が多い. 本研究では, ROS のみについて扱う.

2.2 組込み技術導入の利点

前述の通り, ROS のノードを実行するためには原則, Linux 環境が必要である. Linux 環境は汎用 OS であり, 実行されるプロセスが多く, 大規模なカーネルである. このため, 決められた時間内にタスクを終わらせることができる性能である, リアルタイム性能を確保することが難しい. このことはノードプログラムの応答性低下へとつながり, 意図しない機器の動作を引き起こす可能性がある.

この問題は, システムの一部ノードを, 組込みシステム向けのリアルタイム OS 上で実行することにより解決できると考えられる. リアルタイム OS は, リアルタイム性能を確保することを重視して設計された軽量な OS である. 備える機能が限定的である反面, 実行されるプロセスが少なく, 大規模な汎用 OS に比べてタススケジューリングが容易である. また, スケジューラ自体もリアルタイム性能の確保に向けた設計がなされている, さらに, 汎用 OS は計算資源が限定され, 消費電力およびコストが抑えられた組込み機器に搭載することが可能である. このため, リアルタイム OS を搭載する組込み機器上で ROS のシステムの一部のノードを実行することにより, そのノードアプリケーションの応答性を向上させることが期待される.

また, Linux は大規模なソフトウェアプログラムであるため, 搭載にあたり要求される計算資源の水準が高い. このため, 消費電力, メモリ, および計算能力などの計算資源が豊富であるデバイスを採用する必要がある. ROS を用いて分散システムを構築する場合, 機能が少なく, 要求される計算能力も高くはないノードを, ある機器の上で実行し, ホストと通信を行わせるという構成にすることも考えられる. そのようなノードを Linux の搭載可能な高機能デバイスで実行すれば, 高水準の計算資源がほとんど使われず, かかる消費電力およびコストが無駄になってしまう.

この問題は, ノードの機能およびシステムの要求に合わせ, 計算資源が限定的な組込み機器を採用することにより解決できると考えられる. リアルタイム OS 上の ROS ノード実行環境を活用することにより, Linux が搭載できない機器も活用することができるようになり, ROS ノードを実行する機器の選択肢が多くなる. このため, ノードの実行に要求されるものにあわせて, ノードを実行する機器を選択し, 計算資源を縮小することができる. これにより, システムが利用する計算資源を, より適したものにすることができる.

2.3 組込み技術導入時の課題

ROS のシステムへ組込み機器を導入するメリットを前節において述べた. 一方, 組込み技術を実際に導入するにあたっては, 得られるメリットの他, 被るデメリット, および, システム要件を満たす可能性を検討することが必要となる.

検討材料としてまず, 組込み機器および PC 間の通信速度が考えられる. ROS を活用するシステムにおいて, メッ

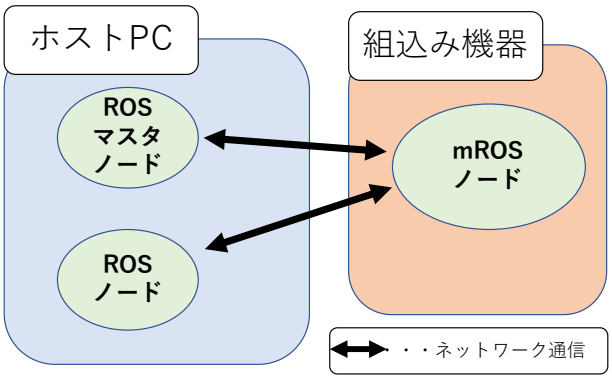

図 1 mROS を用いて構築するシステムの図

セージの通信には通常 TCP が用いられる．このため，リアルタイム OS を採用するとしても，ネットワーク通信も含むリアルタイム性能の確保は非常に難しい．この上でシステムの性能を正確に評価するためには，通信時の特性を把握することが重要である．ここでいう通信時の特性とは，通信にかかる遅延時間およびその揺らぎ，メッセージの到達率，およびそれらとメッセージサイズの関係が考えられる．このような性能について定量的な評価値が存在することにより，ネットワーク通信を含むノードアプリケーションの応答性能について，より正確な見積もりが立てられるようになる．

続いて，実行環境のプログラムサイズが考えられる．組込み機器においてプログラムを実行する場合，機器の持つメモリがプログラムを格納するのに十分な容量を持っていることが必要である．プログラムに機能性を求めると，サイズが肥大化してしまう．一方，サイズを縮小しようとすると，開発効率および機能性に問題が生じる．このため，目的および採用するハードウェアに応じ，適切なプログラム構成を選択しなければならない．比較対象の実行環境についてプログラムサイズを計測し比較することにより，各実行環境の搭載に必要なメモリ容量を見積もることができ，デバイスおよび実行環境の選定を容易に行うことができるようになる．

以上の問題点から本研究では，通信性能およびプログラムサイズについて定量的な評価を行う．

3. 評価対象

3.1 mROS

mROS[1] [2] は，2018 年から著者らが開発している，ROS ノードの軽量実行環境である．mROS は，リアルタイム OS である TOPPERS/ASP カーネル [5]，および，TCP/IP プロトコルスタックを搭載可能な，ミッドレンジの組込み機器を対象としている．現在は，ARM Cortex-A9 のプロセッサを搭載した，GR-PEACH をターゲットボードとして開発が進められている．

mROS を利用したシステムの構成を，図 1 に表す．mROS

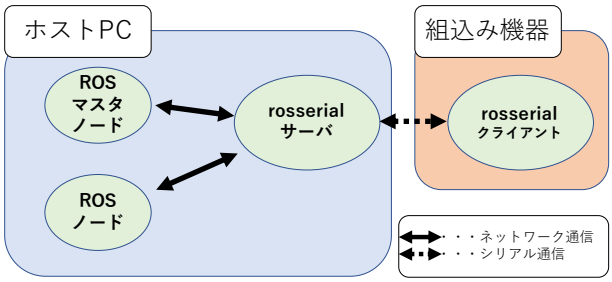

図 2 rosserial を用いたシステム

は，ROS ノードとの通信を実現する mROS 通信ライブラリを，ASP カーネルの上で提供している．これを利用することにより，ROS のノードと通信を行うプログラムを，mROS アプリケーションとしてリアルタイム OS 上で実行することができる．特徴として，図 1 に示す通り，mROS アプリケーションはブリッジを介さず，ネットワークで接続された PC 上で実行される ROS ノードと直接通信を行うことができるという点が挙げられる．，通信の際にブリッジノードを挟む必要がないため，ブリッジノードを挟む場合に比べて通信路の遅延が小さくなる．また，mROS 通信ライブラリは ROS の通信ライブラリと互換性の高い設計をされている上，ユーザが独自定義するメッセージ型を用いる通信にも対応している [4]．このため，既存の ROS ノードプログラムを容易に mROS アプリケーションに移植することもできる．開発言語は C++が利用可能である．

3.2 rosserial

rosserial[3] は，シリアル通信により ROS のノードと通信を行うためのプロトコル，および，その通信を実現するソフトウェアライブラリである．Arduino や STM32 など，シリアル通信の機能を持つ比較的省リソースな組込み機器を対象としている．rosserial を用いた，組込み機器とホスト PC との通信を表す図を図 2 に示す．rosserial を用いる場合，組込み機器で実行するプログラムには，rosserial により提供される ROS 通信ライブラリを用いて通信動作を記述する．一方，組込み機器とシリアル通信を行う PC 上に，ブリッジノードである `rosserial_server` を起動する．この `rosserial_server` が ROS のネットワーク通信とシリアル通信を仲介する．ROS のノードから組込み機器へと送信されるメッセージは，まず `rosserial_server` へと送信され，続いてその内容がシリアル通信で組込み機器上のプログラムへと送信される．反対に，組込み機器から ROS ノードへと送信されるメッセージは，組込み機器から `rosserial_server` へとシリアル通信で送信され，その内容がネットワーク通信で宛先のノードへと送信される．`rosserial_server` は Python，C++，および Java など，複数の実装が用意されている．公式のドキュメント [3] では，Python 実装（`rosserial_python`）の利用が推奨されている．一方，C++実装の `rosserial_server` は，Python 実装

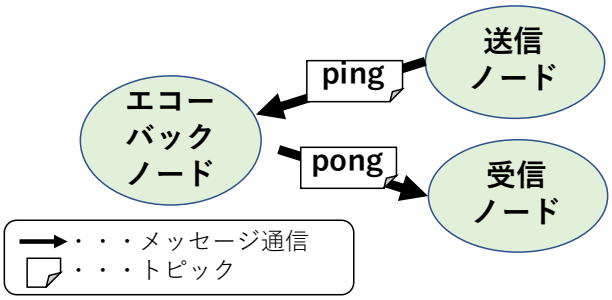

図 3 通信速度の評価に用いるシステムの構成

と比べると機能が制限されるものの, パフォーマンスが高いとされている.

4. 評価

本章ではまず, 本研究で行う評価に関して, 評価内容および評価環境を説明する. 続いて, 各評価項目に関してその評価手法および評価結果を述べ, 結果について議論する.

4.1 評価環境

本節では, 本章で述べる評価を行う環境について説明する.

評価対象の環境は, mROS および rosserial の 2 つとする. 両環境とも, ホストとしてノート PC を用意し, その上で ROS マスタノードおよび評価用の ROS ノードを実行する. ノート PC の環境は以下の通りである.

- Core i7 4500U (1.8GHz)
- Ubuntu 16.04 LTS
- ROS Kinetic

また, 組込み機器には GR-PEACH[6] を使用する. 環境は, 以下の通りである.

- Cortex-A9（400MHz）
- TOPPERS/ASP Kernel 1.9.2
- mROS v1.1 および v1.3
- rosserial(jade-dev)[*1]

実験時は, ノート PC および GR-PEACH を, イーサネットケーブルによりルータを介して同一ネットワークに接続する. 当該ネットワークには, ノート PC および GR-PEACH の 2 機器のみが接続された状態とする. また, rosserial 使用時のシリアル通信のボーレートは 115200baud とする.

4.2 通信遅延

それぞれの実行環境を用いてメッセージの送受信を行なった場合に発生する, 通信遅延時間を測定する.

4.2.1 評価方法

測定に用いるシステム構成を表す図を図 3 に示す. 図に示す通り, システムは送信ノード, 受信ノード, およびエコーバックノードの 3 ノード, および, ping および pong の 2 トピックにより構成する. システムの動作は, まず送信ノードが ping トピックに出版し, エコーバックノードに向けてメッセージを送信する. メッセージを受信したエコーバックノードは即座に同内容のメッセージを pong トピックに出版し, 受信ノードに向けてメッセージを送信する. 評価する通信遅延時間は, 送信ノードからメッセージの送信を開始する際の時刻, および, 受信ノードがメッセージを受信した直後の時刻の差分とする. すなわち, 送信ノードからホスト PC にメッセージを送信して, それが返送されるまでにかかるラウンドトリップ時間（RTT）を測定対象の時間とする.

実際の評価においては, 図 3 に示すシステムを実現するプログラムを実行環境ごとに実装して行う.

mROS の評価において用いる実装を表す図を, 図 4 に示す. 図に示す通り, mROS においては, 送信ノードおよび受信ノードに相当するタスクを組込み機器上で実行する. それぞれのタスクは別個のノードとして, ホスト PC 上のエコーバックノードと直接通信を行う. 処理のフローとしては, （1）メッセージ送信直前に開始時刻を取得し, （2）エコーバックノードに向けてメッセージを送信し, （3）エコーバックノードから帰ってくるメッセージを受信し, （4）受信直後に終了時刻を取得する, というものである.

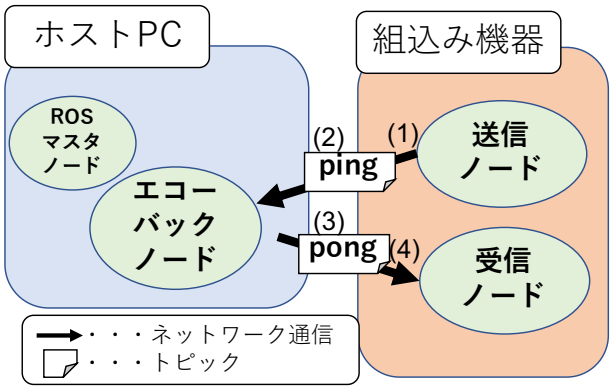

図 4 mROS の通信性能評価に用いるシステム構成

rosserial の評価において用いる実装を表す図を, 図 5 に示す. rosserial を利用する場合, 先述の通り組込み機器上で実行されるアプリケーションは, rosserial サーバを介し

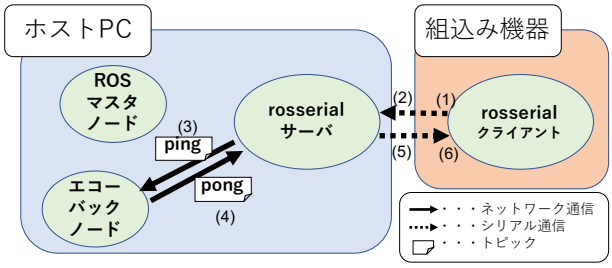

図 5 rosserial の通信性能評価に用いるシステム構成

[*1] rosserial の公式ドキュメントには, Kinetic では jade-dev ブランチのバージョンを使用するよう記載されている.

て通信を行うこととなる．このため, 処理のフローとしては，（1）メッセージ送信直前に開始時刻を取得し，（2）まずシリアル通信でホスト PC 上の rosserial サーバノードにメッセージを送信する．次に，（3）rosserial サーバによりエコーバックノードに向けメッセージが送信され，（4）エコーバックノードより返されたメッセージを rosserial サーバが受信し，（5）シリアル通信によって rosserial クライアントノードに送られる．そして，（6）クライアントノードがメッセージをシリアル通信により受信した直後に終了時刻を取得する．

時刻の取得には, 両環境ともにおいて, TOPPERS/ASP カーネルにより提供される `get_utm` メソッドを用いる．`get_utm` は, 性能評価用のシステム時刻をマイクロ秒単位で取得できるメソッドである．送信ノードから送信されるメッセージのデータは, 1B から 512B まで, 試行ごとに 2 の冪乗で増大するサイズを持ち, 1 試行あたり 200 メッセージを 1 秒間隔で送信されるようにする．なお, 上述および後述の評価結果におけるサイズは, 送受信されるパケット全体のサイズではなく, 送受信データとしてメッセージに含まれる文字列のサイズを指す．なお, 計測結果に影響を及ぼさないため, 評価中は GR-PEACH からホスト PC へのシリアルによるデバッグ出力を一切行わないよう設定した．

4.2.2 評価結果

評価を行なった結果を図 6 および図 7 に示す． 図 6 に mROS の v1.1, rosserial_python, および rosserial_server の結果を, 図 7 に mROS の v1.3 での測定結果を示す．mROS の v1.3 での測定結果は他の環境に比べ極端に計測値が大きかったため, 図 7 に分割した．いずれの図も, 縦軸に計測した通信時間（RTT）を μ 秒単位で, 横軸に送信した文字列のサイズをバイト単位で示している．

また, それぞれの思考における遅延時間の平均値を, 表 1 および表 2 に示す．

表 1 1B-8B 文字列利用時の通信遅延時間の平均値 [μsec]

	1B	2B	4B	8B
mROS v1.1	911.24	909.96	929.03	931.795
rosserial_python	8006.07	7689.52	7878.945	8674.46
rosserial_server	7127.55	8007.625	9290.075	10653.095

表 2 16B-256B 文字列利用時の通信遅延時間の平均値 [μsec]

	16B	32B	64B	128B	256B
mROS v1.1	941.885	921.84	948.325	993.9009901	1024.83
rosserial_python	10694.67	15892.44	18118.78	30654.24378	
rosserial_server	14291.87	14479.115	20817.87	33303.02985	

なお, mROS においては 512B の, rosserial においては 256B および 512B のメッセージを扱うことができなかったため, その結果は掲載していない．それ以外のサイズのメッセージ送受信に関しては, 全ての環境の全ての試行において, 送信した 200 メッセージ全てを失うことなく受信することができた．

4.2.3 議論

図および表から, mROS の v1.1 では, メッセージのサイズにかかわらおよそ 1ms ほどの遅延時間で通信ができており, なおかつ試行内のばらつきも少ないことがわかる．一方 rosserial では, 少なくとも 5ms ほど時間がかかる上, メッセージサイズに比例して通信時間が増大しており, 試行内でのばらつきも比較的大きいことがわかる．このため, 少なくとも 256B 以下のメッセージに関しては, シリアル通信を用いる rosserial より, ネットワーク通信を用いる mROS の方が, 少ない遅延時間で通信でき, またばらつきも少ないといえる．したがって, この 2 者では mROS の v1.1 の方が, リアルタイム性の確保および通信遅延の見積もりが容易であるといえる．なお, 本研究では rossreial でのシリアル通信のボーレートを 115200baud と設定したが, より高いボーレートを設定することにより通信速度が向上すると期待される．しかし高いボーレートを用いることにより, メッセージのビット誤りの発生確率が高くなり, 正しく受信できないメッセージの割合が増加してしまう可能性もある．

mROS の v1.3 では, 通信にかかる遅延時間が極端に大きく, およそ 20ms から 80ms ほどかかってしまっていることがわかる．測定結果を確認すると, 通信遅延時間が一定の割合で増加および減少を行なっていることが確認できた．これより, 原因を直接特定できなかったものの, mROS の通信ライブラリ, あるいは TCP/IP スタックによるソフトウェア制御が原因で遅延が発生していると考えられる．

rosserial_python および rosserial_server を比較すると, 通信時間の平均値およびばらつきのどちらに関しても, rosserial_python の方が小さくなっていることがわかる．これは, C++実装である rosserial_server の方が性能が優れているという予想に反する結果である．この原因として, rosserial クライアントからシリアル通信で渡されるメッセージの型は実行時に判明するため, C++による実装の方がより複雑な実装を要求されるからという可能性が考えられる．また, rosserial_server の利用により性能向上が期待される場合が, 本評価で用いたものよりさらに大きなメッセージを用いた場合である可能性も挙げられる．

4.3 プログラムサイズ

4.3.1 評価方法

各実行環境について, 組込み機器に搭載する実行可能ファイルのプログラムサイズを比較する．評価対象のファイルは, ビルドによる生成物のうち, プログラム全体を含むファ

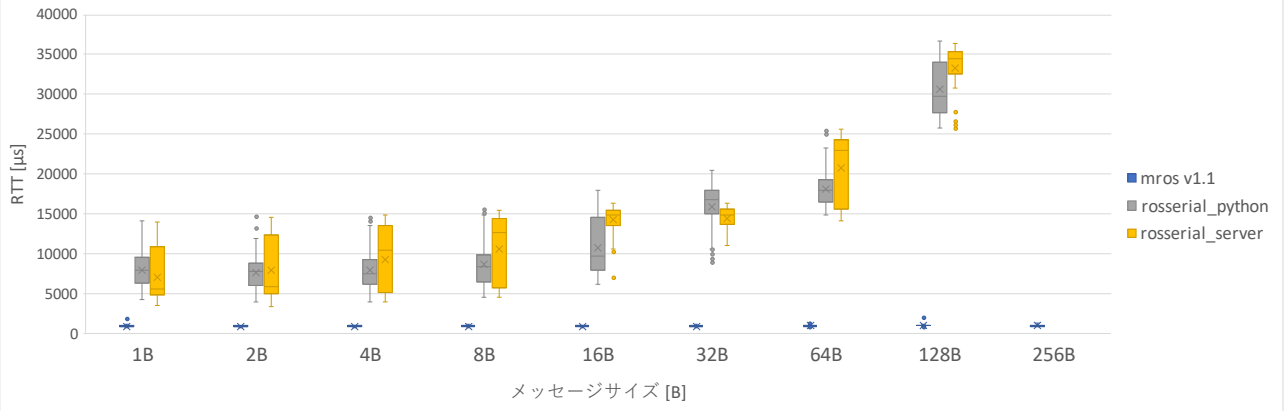

図 6 mROSv1.1, rosserial_python, rosserial_server を用いた場合の RTT

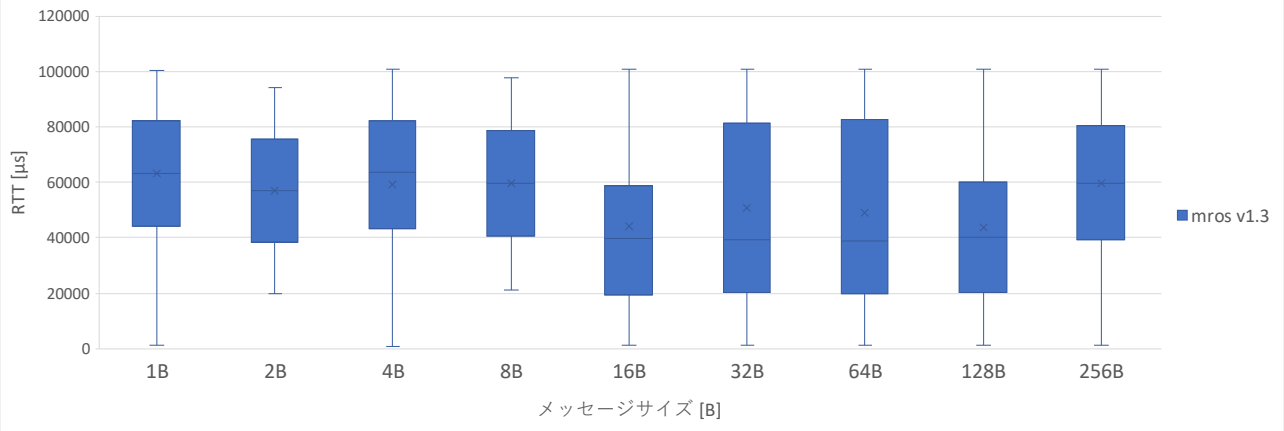

図 7 mROSv1.3 を用いた場合の RTT

イル, および mbed ライブラリのファイルとする. 前者は, `objcopy` コマンドでバイナリファイルに変換されるファイルを利用する. mROS であれば `asp.bin` が, rosserial であれば `asp` がこれに該当する. 後者は, mROS および rosserial の両者において生成される `libmbed.a` を対象とする. このファイルを評価対象とする理由は, TCP スタックがファイルサイズにおよぼす影響を調査するためである. mROS では, 軽量 TCPIP スタックである lwip を採用している. lwip は mbed ライブラリとして提供されているため, mbed ライブラリのファイルに対し影響を与えていると考えた. 評価では Linux の `size` コマンドを用いて, 各ファイルの各セクションごとのプログラムサイズを取得する.

評価は, 通信性能の評価において GR-PEACH に搭載したものと同じアプリケーションのビルド生成物を使用する. ただし, 送受信メッセージのためのバッファサイズに関しては以下のように調整を加えた.

mROS の v1.1 では, `mros_msg-gen/msg_max_size.h` ファイルにおいて定義されている, 出版メッセージの最大サイズの定義のみを変更する. このサイズ定義のファイルは, 文献 [4] に対応し, 出版されるメッセージのサイズに応じて内部メモリのサイズを自動的に変更するために実装されたものである. 仕様上, コンパイル時に長さが定まらない文字列型を利用する場合は, 余裕を持たせるため 1 MB ほどの非常に大きな領域を確保しようとする. このため, この箇所のみ修正し, 後述する rosserial と同じく 512MB の領域を出版データに向けて確保するように設定した. これ以外の, プログラム内部で定義されているメモリ領域については変更しない.

mROS の v1.3 では, `ros_ws/(アプリ名)/mros_config/mros_sys_config.h` において, アプリケーションで利用するメモリプールのサイズが設定できるようになっている. 出版および購読メッセージ向けに分けて確保されている訳ではない上, 非常に大きなサイズを余分に確保している実装ではない. このため, mROS v1.3 のサイズ評価においてはプログラムに変更を加えない.

rosserial では, `ros_lib/ros/node_handle.h` において, 出版および購読メッセージのためのバッファのサイズが定義されている. サイズの大きいメッセージを扱う場合は, この値を変更し大きい領域を確保することが推奨されている. 初期値は出版および購読のためのバッファ領域のサイズがともに 512KB であるが, 評価にあたってはこのサイズは変更しない.

4.3.2 評価結果

`asp.bin` および `asp` について評価した結果を表 3 に,

libmbed.a について評価した結果を表 4 に示す.

表 3　asp.bin および asp のサイズ評価結果

	text[Byte]	data[Byte]	bss[Byte]	dec[Byte]
mROS v1.1	419628	2932	3645156	4067716
mROS v1.3	346564	2876	330632	680072
rosserial	129380	2668	63124	195172

表 4　libmbed.a のサイズ評価結果

	text[Byte]	data[Byte]	bss[Byte]	dec[Byte]
mROS v1.1	266954	52954	45719	365627
mROS v1.3	266954	52954	45719	365627
rosserial	115003	52760	956	168719

4.3.3　議論

プログラム全体のサイズでは, mROS v1.1 がもっとも大きく, およそ 4MB ほどになった. ついで mROS v1.3 が約 680KB, rosserial がもっとも小さく 200KB 弱ほどとなった.

mROS v1.1 では, 画像など大きなデータの送信に対応するため, 通信ライブラリで扱う共有メモリの大きさをかなり大きく確保している. このため, bss セクションのサイズがおよそ 3.6MB ともっとも大きく, プログラム全体のうち大半を占めていまっている. 一方 mROS v1.3 では, 内部実装が整理され, 小規模なデータを主に扱うことを想定したメモリ領域の確保を行うようになった. これにより, bss セクションが大幅に縮小した. text セクションに関して v1.1 から v1.3 で 80KB ほど縮小していることからも, 内部実装が整理されたことがうかがえる.

rosserial および mROS v1.3 を比較すると, rosserial の方がさらに 500KB ほど小さくなっている. rosserial のプログラム全体のサイズは 200KB にも満たないゆえ, これは TOPPERS/ASP カーネルのプログラムも含んでいる. このため, ベアメタルアプリケーションで rosserial を利用することにより, さらに小規模なプログラムにすることができると考えられる. 2 環境の大きさに差がある原因としては, ネットワーク通信を行わないため TCP/IP プロトコルスタックを搭載していないということが挙げられる. このことは, 表??に示す mbed ライブラリプログラムの比較結果からも読み取ることができる. しかし, それでも mROS v1.3 は 1MB を下回るサイズで実現されており, ミッドレンジ帯の組込みシステムであれば余裕を持って搭載することができるともいえる.

5.　結論

本研究では, ロボットシステム開発支援プラットフォームである ROS を活用するシステム開発に, 組込みシステムを導入することについて議論し, 考えられる問題点の解消を目標として組込みシステム上の ROS ノード実行環境について定量的な評価を行なった. 評価は, ネットワーク通信を用いる mROS およびシリアル通信を用いる rosserial を対象とし, 通信性能およびプログラムサイズの移植性の 2 項目について行なった. 評価により, それぞれの性能および特徴を示す結果が得られた. この評価結果が, ROS を用いるシステムへの組込み機器の導入促進に繋がると期待される.

謝辞

参考文献

[1] Takase, H., Mori, T., Takagi, K. and Takagi, N.: *Work-inProgress: Design Concept of a Lightweight Runtime Environment for Robot Software Components Onto Embedded Devices*, 2018 International Conference on Embedded Software (EMSOFT), pp. 1–3 (2018)

[2] Takase, H., Mori, T., Takagi, K. and Takagi, N.: *mROS: A Lightweight Runtime Environment for Robot Software Components onto Embedded Devices*, HEART 2019 Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, Article No. 7 (2019).

[3] Ferguson, M.: rosserial, http://wiki.ros.org/rosserial.

[4] Yugen, H., Takase, H., Takagi, K. and Takagi, N.:*A functionality expansion of the lightweight runtime environment mROS for the user defined message types*, Proceedings of Asia Pacific Conference on Robot IoT System Development and Platform (APRIS2019), 1-7 (2019).

[5] TOPPERSproject: TOPPERS/ASP kernel, https://www.toppers.jp/ asp-kernel.html.

[6] ルネサスエレクトロニクス: GR-PEACH, http://gadget.renesas.com/ ja/product/peach.html.

正誤表

下記の箇所に誤りがございました．お詫びして訂正いたします．

訂正箇所	誤	正
7 ページ 4.3.3 23 行目	表??	表 4