Vol.2018-ARC-230 No.2
Vol.2018-SLDM-183 No.2
Vol.2018-EMB-47 No.2
2018/3/7

ROS2における通信特性に応じたDDS実装の動的選択機構の実現

森田　錬[1]　松原　克弥[1]

概要：適用範囲の拡大にともなって開発されている ROS2 (Robot Operating System 2.0) は，通信基盤に DDS (Data Distribution Service) 仕様を採用し，システムの目的に応じて DDS 実装を選択できる．現在，複数のベンダが DDS 実装を提供しており，共有メモリ使用の有無など DDS 実装毎に特徴が異なる．現在の ROS2 では，システム毎に一つの DDS 実装しか指定できないため，特徴の異なる通信毎に最適な DDS 実装へ切り替えることができない．本研究では，ROS2 におけるモジュール間通信の性能向上を目的とし，通信特性に応じた DDS 実装の動的選択機構を提案する．本提案機構は，通信開始時にデータサイズや通信範囲などの通信特性を推定し，最適な DDS 実装を自動的に選択する．本稿では，通信性能とオーバーヘッドの観点で現 ROS2 実装との比較評価を行い，DDS 実装を動的に選択することの有意性を示す．

キーワード：Robot Opetaring System, ROS2, DDS, Publish-Subscribe 型通信

A Mechanism of Dynamic Binding a Proper DDS Implementation for Optimizing Inter-node Communication in ROS2

Ren Morita[1]　Katsuya Matsubara[1]

Abstract: Nowadays, ROS2 (Robot Operating System 2.0) has been being developed from scratch in order to support new applications. One of the most significant change in ROS2 is to follow the DDS standard specification as an interface of the inter-module communication. Currently some of companies and OSS communities provides DDS implementations which have different characteristics such as using the shared memory. The current implementation of ROS2 allows to specify only one DDS implementation to be used in a system through application code or the environment variable at running, nevertheless each communication may have different characteristics and each DDS implementation may be optimized for different communication types. This research proposes a mechanism to allow multiple DDS implementations existing in a system and to dynamically bind one into each communication. The proposed mechanism can estimate communication area, maximum data size, requirement of the QoS control, for each communication channel and then it can select a proper DDS implementation. This thesis shows comparison results of performance and overhead of the current available DDS implementations to evaluate effectiveness of this proposal, describes implementation of the mechanism.

Keywords: Robot Operating System, ROS2, DDS, Publish-Subscribe Messaging

1. はじめに

制御用ソフトウェアは，複数の機能モジュールで構成された分散システムとして実装することが多く，通信性能やリソース管理，スケーラビリティなどの制約を解決するために専用のフレームワークが利用されることが多い．ROS (Robot Operating System) は，Open Source Robotics Foundation(以下，OSRF) が提供するオープンソースのロボット制御用フレームワークである [1], [2]．ROS を用いて開発する制御ソフトウェアは，システム全体をノードと

[1] 公立はこだて未来大学 システム情報科学部
School of Systems Information Science, Future University Hakodate

Vol.2018-ARC-230 No.2
Vol.2018-SLDM-183 No.2
Vol.2018-EMB-47 No.2
2018/3/7

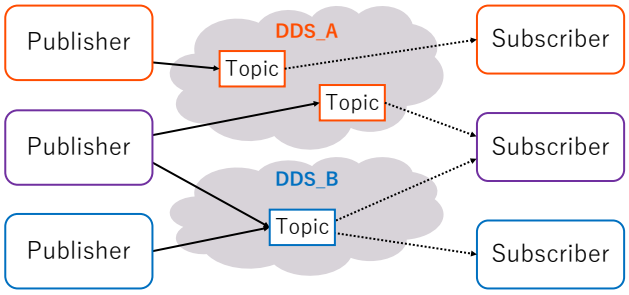

図 1　提案機構における DDS 実装選択

呼ばれる小さな機能モジュールに分割し，それらが互いにデータ通信を行うことで実現される．ROS におけるノード間の通信のほとんどは，データ中心指向の非同期的通信が可能な Publish-Subscribe (以下，Pub/Sub) 型通信によって行われる．Topic と呼ばれるメッセージのクラス分けを介した Pub/Sub 型通信を行うことで，入力側と出力側の頻繁なデータ共有および動的でスケーラビリティの高いネットワーク構成が実現できる．

ROS2 は，従来の ROS (以下，ROS1) では想定されていなかった複数ロボットや小型の組込みシステム，リアルタイム処理，低品質なネットワークなどでの利用に限界があったため，新たなユースケースへの対応を目的として，2017 年 12 月に正式リリースを公開し，現在も開発が続いている [3]．ROS2 における最大の変更点の一つは，通信基盤に Object Management Group が策定した標準仕様である DDS (Data Distribution Service) を採用したことである [4]．複数のベンダが仕様に沿って開発した DDS 実装を提供しており，それぞれ共有メモリ使用や省メモリフットプリントなどの特徴を持つ．現 ROS2 実装では，システムの目的に応じて，実装時または実行時に DDS 実装を一つだけ選択することができる．そのため，ロボット制御システムでは，映像データやセンサデータ，制御命令などノード間の特性が異なる通信毎に最適な DDS 実装を選択することができない．

本研究では，ROS2 におけるノード間通信の最適化を目的とし，ノード間通信の特性に応じて最適な DDS 実装を動的に選択する機構を提案する (図 1)．提案する機構では，通信時に指定される Topic 毎に推定した通信特性を応じて最適な DDS 実装を選択する．Topic ごとに最適な DDS 実装を利用するために，DDS 実装のロードを Topic 指定時まで遅延する．通信時は，対応する Topic に応じて最適な DDS 実装を選択し，動的な切り替えを行う．これらの機能を実現した DDS 実装の動的選択機構によって，ノード間の通信の最適化を実現する．

2. 既存 DDS 実装の特性評価

複数のベンダが DDS の仕様に沿った開発を行っており，現在 ROS2 がサポートしている DDS 実装として，eProsima 社の FastRTPS[5]，Real-Time Innovations 社の Connext[6]，PrismTech 社の OpenSplice[7]，OSRF が実装している FreeRTPS[8] などがある．本章では，DDS 実装の性能比較・評価に関する既存研究の結果を紹介し，さらに，追実験として行った FastRTPS と OpenSplice の比較実験の結果を示す．複数のベンダが提供する DDS 実装は，データサイズと通信範囲，再送確認の有無といった観点でそれぞれ特徴を持ち，DDS 実装を切り替えることに大きな利点があることが示された．

2.1 関連研究における評価

2.1.1 ROS2 における DDS 実装の性能評価

Maruyama らは，ROS2 の通信性能とオーバーヘッドの調査を行い，ROS1 との性能比較を行った [9]．ROS2 における調査では，FastRTPS と Connext，OpenSplice の三つの DDS 実装を利用した ROS アプリケーションを対象として，メモリ消費量やスレッド，レイテンシ，スループットについての比較，調査を行っている．メモリ消費量やスレッド数の評価では，FastRTPS が最も軽量であるという結果が示された．レイテンシの評価では，OpenSplice が最も小さく，スループットの評価では，Connext が最も良い結果となった．

2.1.2 OpenSplice と Connext の性能比較

Bellavista らは，OpenSplice と Connext を対象として DDS 実装の性能比較を行った [10]．OpenSplice と Connext のシステムアーキテクチャの方式の違いについて調査し，単位時間当たりのデータ送信数とスループット，ラウンドトリップタイムについて比較を行っている．

結論として，OpenSplice は，非常に高い頻度のメッセージ送信によって，高いオーバーヘッドコストが発生してスケーラビリティが妨げられるものの，小さいデータを送信する性能が高いという評価が示された．対して，Connext は，小さいデータにおける通信性能が OpenSplice に劣る反面，メッセージのフラグメントやメモリ管理機能の性能が高く，スケーラビリティが高いことが示された．

2.2 FastRTPS と OpenSplice の比較実験

追実験として，現在 ROS2 で利用可能な FastRTPS と OpenSplice を対象として，再送確認の有無を切り替えながらスループットとレイテンシを計測して評価した．加えて，実行時のメモリ消費量およびスレッド数を計測した．実験環境を表 1 に示す．実験に用いた ROS アプリケーションは，単一コンピュータ内の別々のプロセスに Publisher

Vol.2018-ARC-230 No.2
Vol.2018-SLDM-183 No.2
Vol.2018-EMB-47 No.2
2018/3/7

表 1　実験環境

CPU	Intel Corei3-3110M 2.4GHz
Memory	4GB
Kernel	Linux 4.4.0-103-generic
Distribution	Ubuntu 16.04.3 LTS
ROS2	Ardent Apalone
FastRTPS	1.5.0
OpenSplice	6.7.0+osrg2-2 xenial

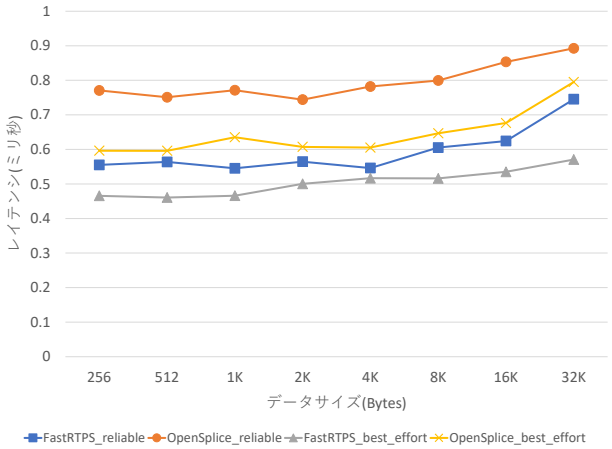

図 2　小さいデータにおける FastRTPS と OpenSplice のレイテンシ

表 2　ROS2 における FastRTPS または OpenSplice 実行時の仮想メモリと物理メモリ，スレッド数

	仮想メモリ (KB)	物理メモリ (KB)	スレッド
FastRTPS	460744	70052	7
OpenSplice	1871840	43940	28

と Subscriber のノードを起動し，その二つのノード間で一つの Topic を指定して通信を行う．通信するデータサイズは 256B から 4MB まで変化させ，各サイズにおいて 1 秒間に 10 回 Publish する通信を行った．レイテンシの計測は，Publisher がデータを Publish してから Subscriber が Subscribe するまでの時間を 100 回計測した結果を平均した．スループットは，DDS 実装のワイヤプロトコルである RTPS プロトコルをキャプチャし，1 秒間当たりの送信データ量を計測した．実行時のメモリ消費量およびスレッド数の計測は，レイテンシおよびスループットの計測に利用した ROS アプリケーションの 256B のデータを送信するノードに対して，仮想メモリと物理メモリ，動作しているスレッド数を計測した．

FastRTPS と OpenSplice のレイテンシの計測結果を図 2 と図 3 に示す．256B から 32KB までの小さいデータでは，再送確認の有無に関わらず，FastRTPS よりも OpenSplice のほうがレイテンシが小さい．しかしながら，64KB から 4MB までの大きいデータでは，OpenSplice では再送確認の有無によるレイテンシの変化がほとんどないが，FastRTPS では，再送確認の有無によってレイテンシに大きく差がある．再送確認がある場合は OpenSplice，再送確認がない場合には FastRTPS の方がレイテンシが小さい．なお，スループットの計測結果については，FastRTPS と OpenSplice のどちらも，データサイズ，再送確認の有無によって変化が無かった．

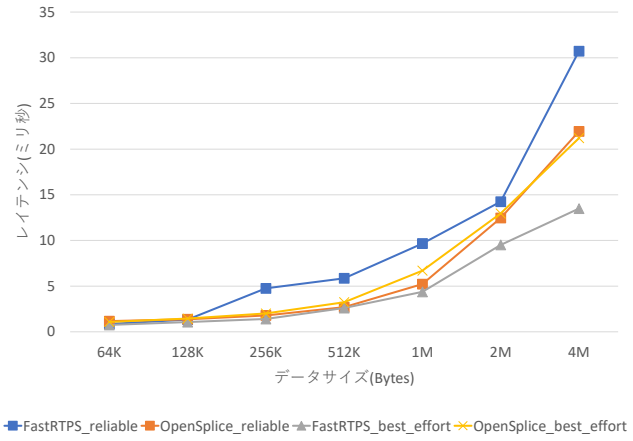

図 3　大きいデータにおける FastRTPS と OpenSplice のレイテンシ

FastRTPS と OpenSplice のメモリ消費量とスレッド数の計測結果を表 2 に示す．FastRTPS と比較して，OpenSplice の仮想メモリが約 4 倍消費されており，スレッド数も 4 倍のスレッドが作成されていた．

本実験結果は，レイテンシの観点では，FastRTPS を利用するべきであるが，通信するデータサイズが大きく再送確認を行う場合には OpenSplice が適していることを示している．また，メモリ消費量とスレッド数の比較では，FastRTPS の方が計算資源を節約できることを示す結果が得られた．

3. DDS 実装の動的選択機構

最適な DDS 実装を選択し，通信時に DDS 実装を動的に切り替える機構を実現するためには，以下の技術的課題を解決する必要がある．

第一の課題は，ノード間通信の特性の推定である．ノード間の通信は，通信時に指定される Topic によって通信する範囲や通信するデータサイズ，再送確認の有無など様々な側面に関して特性がある．しかし，Topic 自体は Publish，Subscribe をする際に利用したいデータのクラス分けの名前としての役割しか持っていない．ノード間通信毎に最適な DDS 実装を選択するために，ノード間通信の特性の推定するための情報が必要となる．

第二の課題は，DDS 実装をロードするタイミングの変更である．現 ROS2 の実装では，DDS 実装のロードはシス

Vol.2018-ARC-230 No.2
Vol.2018-SLDM-183 No.2
Vol.2018-EMB-47 No.2
2018/3/7

テム起動後に最初に呼び出される DDS 実装の初期化 API が呼び出されるタイミングで行われる (図 4(a))．ロードした DDS 実装を利用して初期化が完了した後，指定された Topic の登録と通信が行われる．しかしながら，提案する機構では，指定された Topic を登録する際に利用する DDS 実装が決定するため，システム起動時に利用する DDS 実装をロードすることはできない．したがって，Topic を登録するタイミングまで DDS 実装のロードを遅延する必要がある．

第三の課題は，DDS 実装の動的切替である．本提案では，Topic 毎に利用する DDS 実装が選択されるため，複数の DDS 実装をシステム実行中に切り替えなければならない．そのためには，DDS 実装の API が呼ばれるタイミングで，最適な DDS 実装の選択結果から通信に利用する DDS 実装を動的に切り替える必要がある．

以降の節では，前述のそれぞれの課題に対して実現する機能について述べる．

3.1 最適な DDS 実装の選択ポリシ

Topic 毎のパラメータ設定，および，各パラメータと DDS 実装の対応付けをノード構成に基づいて設定することで，最適な DDS 実装を選択する．複数の側面から特徴づけられる DDS 実装とノード間通信の特性を対応付けるために，通信に指定される Topic 毎にパラメータを設定する．本提案では，前章で行った追実験の結果から示された FastRTPS と OpenSplice の特徴と対応付けることが可能なパラメータを検討し，以下の三つとした．

データサイズ

large_data　64KB 以上のデータの通信

small_data　64KB より小さいデータの通信

通信範囲

intra　単一プロセス内でスレッドとして動作するノード間のプロセス内通信

local　単一コンピュータ内のノード間のプロセス間通信

remote　異なるコンピュータのノード間のプロセス間通信

再送確認

reliable　再送確認を行う QoS 設定

best_effort　再送確認を行わない QoS 設定

これらの三つのパラメータの組み合わせと DDS 実装との対応付けをマトリクスとして設定することで最適な DDS 実装を選択することができる．マトリクスの例を表 3 に示す．この例では，デフォルトの DDS 実装を DDS_A とし，large_data と remote と reliable が設定されている Topic では DDS_B，large_data と local または large_data と remote のパラメータが設定されている Topic には DDS_C，best_effort が設定されている Topic は DDS_D を利用するように，パラメータの組み合わせと DDS 実装の対応付けを設定した．それらの対応付けを，DDS_B > DDS_C > DDS_D の順に優先度を設定したマトリクスを示している．

表 3　パラメータの組み合わせと DDS 実装の対応例

再送確認	通信範囲 / データサイズ	intra	local	remote
best_effort	small_data	DDS_D	DDS_D	DDS_D
	large_data	DDS_D	DDS_D	DDS_D
reliable	small_data	DDS_A	DDS_C	DDS_C
	large_data	DDS_A	DDS_C	DDS_B

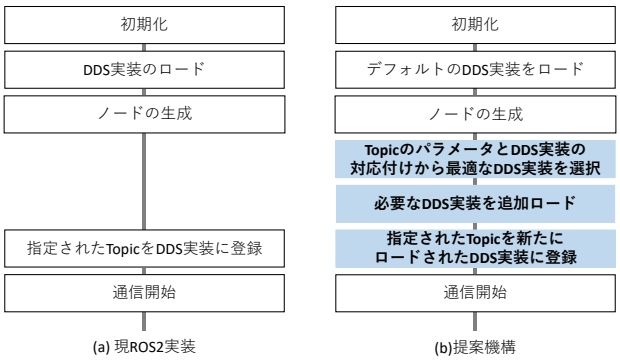

図 4　DDS 実装のロード処理におけるフロー

3.2 DDS 実装のロードタイミングの遅延

ノード間通信の特性と DDS 実装の対応付けに応じて複数の DDS 実装を利用するために，Topic を指定するタイミングで対応付けられた DDS 実装をロードする．前節で述べたパラメータは Topic 毎に設定されているため，Topic を指定するタイミングで利用する DDS 実装が決定する．Topic の指定時に DDS 実装をロードすることで，利用されない不要な DDS 実装のロードを回避することができる．

図 4 に DDS 実装をロードする処理フローを示す．現 ROS2 の実装では，システム起動直後の初期化を行った後，DDS 実装のロードを行う．その後，ノードが生成され指定された Topic を DDS 実装に登録し通信が行われる．本提案では，指定された Topic をデフォルトの DDS 実装に指定する前に，Topic のパラメータと DDS 実装の対応付けから最適な DDS 実装を選択し，必要な DDS 実装を追加してロードする．そして，指定された Topic を追加ロードされた DDS 実装に登録して通信を開始する．選択された DDS 実装が既にロードされている場合は，追加のロードを行わず，選択した DDS 実装への Topic の登録を行う．

3.3 Topic に応じた DDS 実装の動的切替

Topic 毎に DDS 実装を選択できるようにするために，Publish 毎に DDS 実装を動的に切り替える．DDS の API は仕様によって統一されており，ROS クライアントライブ

Vol.2018-ARC-230 No.2
Vol.2018-SLDM-183 No.2
Vol.2018-EMB-47 No.2
2018/3/7

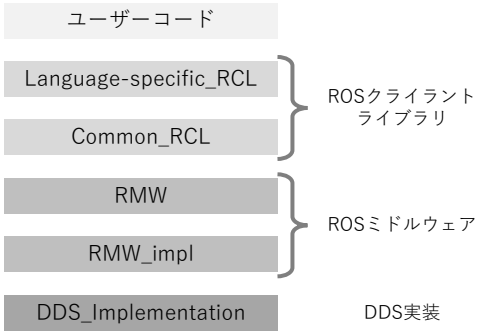

図 5 ROS2 のシステムアーキテクチャ

ラリからは同一のインタフェースで呼び出されている．一つのノードが通信毎に異なる Topic を指定ができるため，それぞれの Topic に異なる DDS 実装が対応付けられる場合がある．したがって，DDS の通信用 API が呼ばれた際，Topic と DDS 実装の対応付けをその都度確認し，利用する DDS 実装を切り替える．そのほか，ノードの名前の取得やノード数のカウントといった Topic に依存しない API は，呼び出し元のノードが利用する DDS 実装に応じて呼び出す DDS 実装を切り替える．

4. 実装

ROS2 のシステムアーキテクチャを図 5 に示す．ROS2 のシステムアーキテクチャは，ROS クライアントライブラリ，ROS ミドルウェア，DDS 実装によって構成される．ROS クライラントライブラリは，ユーザーコードから ROS の機能を実行する API を提供し，対応するプログラミング言語毎に特有の機能を提供する言語依存部と，ノードの一般的な機能を提供する共通部に分けられる．ROS ミドルウェアは，実行時に指定された DDS 実装に対応する DDS 依存部 (RMW_impl) と，ROS クライアントライブラリから DDS の API を利用するために抽象化されたインタフェースを提供する ROS ミドルウェア共通部 (RMW) に分けられる．DDS 実装は，RMW_impl によって，実行時または実装時に設定された環境変数 *RMW_IMPLEMENTATION* から利用する DDS 実装を特定した後，共有ライブラリとして静的にロードされる．ただし，ROS2 を利用したアプリケーションの実装方式としては，一つのプロセスに複数のノードをスレッドとして実行する実装が推奨されており，プロセス内に存在するノード間の通信にはロードした DDS 実装が利用されておらず，ROS クライアントライブラリの共通部分がプロセス内通信を行っている．

提案する DDS 実装の動的選択機構は，DDS 実装のロードと API の抽象化を行う RMW に機能を追加することで実現する．ROS ミドルウェア内で 3.1 節で述べた通信特性を示すパラメータを取得し，ROS クライアントライブラリからの呼び出し時に DDS 実装のロードと Topic 毎に利用する DDS 実装の切替を行う．加えて，データサイズのパラメータの取得のために，Topic の指定に合わせて予想データサイズを設定できるように ROS クライアントライブラリを拡張する．通信範囲と再送確認のパラメータは，それぞれ IP アドレスおよびプロセス ID，DDS に設定する QoS プロファイルの内容から自動設定する．

Topic 毎のパラメータの設定およびパラメータと DDS 実装の対応付けを行うために，最適な DDS 実装の選択が可能となる設定ファイルを読み込む機能を実装した．設定ファイルには，ノード間通信時に指定される Topic 毎のパラメータおよびパラメータの組み合わせと DDS 実装の対応付けの情報を XML 形式で記述し，システム起動時にそれらを参照することでパラメータの値と DDS 実装の対応付けを取得することで，最適な DDS 実装の選択を実現した．一つのノードが行う複数の通信毎に異なる Topic を指定し，それぞれが違う DDS 実装を選択する場合に備えて，複数の DDS 実装をロードする機能と Topic に応じて特定の DDS 実装の抽象化を行う ROS ミドルウェアの実行部分を動的に切り替える機能を実装した．各ノードは，起動時にデフォルトの DDS 実装をロードし，その後必要に応じて，DDS 実装を追加してロードすることで複数 DDS 実装のロードを実現した．また，複数の DDS 実装がロードされた場合に，ノードが Publish を行うときに，Topic と DDS 実装の対応付けをその都度確認し，利用する DDS 実装をその都度選択することで動的な DDS 実装の切り替える機能を実装した．

表 4 FastRTPS および提案機構の実装の仮想メモリと物理メモリ，スレッド数

	仮想メモリ (KB)	物理メモリ (KB)	スレッド
FastRTPS	460744	70052	7
提案機構の実装	2201492	99068	34

5. 実験・評価

5.1 概要

実装した機構を用いた ROS アプリケーションを対象として，レイテンシと消費メモリおよびスレッド数の比較調査を行った．2.2 節で述べた追実験の結果によって，OpenSplice が適していることが示された，大きいデータの通信と再送確認を行う場合において，現 ROS2 実装で FastRTPS を用いたケースと実装した機構によって最適化したケースを比較し考察を行う．そのほかの条件については追実験と同様である．

5.2 結果と考察

レイテンシの計測結果を図 6 に示す．提案機構の実装を用いることで，現 ROS2 の実装で OpenSplice を用いた場合とほぼ同じレイテンシとなった．レイテンシは，提案機

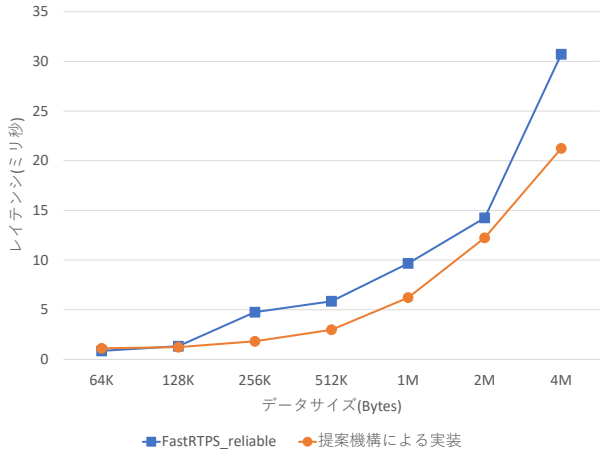

図 6 大きいデータの通信と再送確認を行う場合における FastRTPS と提案機構の実装のレイテンシ

構による最適な DDS 実装の選択により，ノード間通信が最適化されたことで，本稿の提案機構の有意性を示すことができた．

表 4 に FastRTPS および提案機構の実装の仮想メモリおよび物理メモリ，スレッド数を示す．提案機構の実装においては，メモリ消費量，スレッド数ともに，FastRTPS と OpenSplice の合算と同程度となった．メモリ消費量とスレッド数については，ロードする DDS 実装の数に比例して増加すると考えられるため，改善が必要である．例えば，intra のみの通信であれば DDS 実装のロード自体が不要であるため，どの DDS 実装もロードしない機構の実現などが考えられる．

6. まとめ

本研究では，ROS2 におけるノード間通信の最適化が期待できる通信特性に応じて最適な DDS 実装の動的選択機構を実現した．ノード間通信の特性から最適な DDS 実装を選択し，システム実行時に複数の DDS 実装の中から動的な切り替えを行うための技術的課題を検討し，課題に対する解決策を提案した．ノード間通信の特性の推定には，ノード間通信時に指定される Topic 毎にデータサイズ，通信範囲，再送確認の三つのパラメータを設定することで解決し，パラメータの組み合わせと DDS 実装を対応付けることで最適な DDS 実装の選択を可能とした．動的な DDS 実装の切り替えには，DDS 実装のロードを Topic の指定時まで遅延して複数の DDS 実装をロードし，通信時に指定される Topic に応じて利用する DDS 実装をその都度切り替えることで解決した．プロトタイプ実装を行い，二つの DDS 実装を用いた比較評価を行った．パラメータに応じて動的に切り替えることで通信性能の向上が確認できた．一方，複数の DDS 実装をロードすることによるメモリフットプリントの増加が課題となった．

今後の課題として，以下の四つがある．一つ目は，複数の Subscribe を行う際の DDS 実装の動的切替の実装である．DDS 実装を動的に切り替えて Subscriber するノードを作成することは可能であるが，複数の DDS 実装を利用するノードに対して別々の Topic を並列して受信状態することができていない．現 ROS2 の実装を更に調査して，解決策を講じる必要がある．二つ目は，メモリフットプリントの課題解決である．本提案機構による実験の結果，複数の DDS 実装をロードすることによる，レイテンシの低下とメモリフットプリントの増加がトレードオフの関係であることが示された．新たに，通信性能またはメモリフットプリントの優先度が設定可能なパラメータの追加といった解決策が必要である．三つ目は，Topic 毎のパラメータの設定の自動化である．プロトタイプでは，Topic 毎のパラメータを事前に設定ファイルに記述することで行うが，IP アドレスやプロセス ID，QoS プロファイル，送信するメッセージの大きさなどから自動的にパラメータを取得するための実現方法を検討する必要がある．四つ目は，ノード構成の変化による DDS 実装の動的な再切替である．現在の実装では，事前に Topic 毎のパラメータを設定ファイルに記述している．そのため，システムの実行途中にノード構成が変化した場合，状況の変化に対応して利用する DDS 実装の切替を行うことができない．ノード数の変化や Publish の停止など動的なノード構成に対応して，Topic 毎のパラメータを変更し，パラメータの変化によって利用する DDS 実装を切り替える手法を検討する必要がある．

参考文献

[1] ROS.org, `http://www.ros.org/`(accessed December 17 2018).
[2] M. Quigley et al., “ROS: an open-source Robot Operating System,” in ICRA workshop on open source software, 2009, vol. 3, p. 5.
[3] ROS2, `https://github.com/ros2`(accessed December 17 2018).
[4] Data Distribution Services, `http://portals.omg.org/dds/`(accessed December 17 2018).
[5] eProsima Fast-RTPS, `http://www.eprosima.com/index.php/products-all/eprosima-fast-rtps`(accessed December 17 2018).
[6] Real-Time Innovations RTI Connext, `https://www.rti.com/products/dds`(accessed December 17 2018).
[7] PRISMTECH OpenSplice, `http://www.prismtech.com/dds-community`(accessed December 17 2018).
[8] Open Source Robotics Foundation FreeRTPS, `https://github.com/ros2/freertps`(accessed December 17 2018).
[9] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ROS2. Proceeding,” EMSOFT ‘16 Proceedings of the 13th International Conference on Embedded Software, Article No.5, 2016, pp. 110.
[10] P. Bellavista, A. Corradi, L. Foschini, and A. Pernafini, “Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations,” in Computers and Communications (ISCC), 2013 IEEE Symposium on, 2013, pp. 377383.