Vol.2019-CSEC-87 No.4
2019/12/3

ロボットオペレーティングシステムの対応OSによる暗号通信における性能評価

亘理 克好[1,a)]　大久保 隆夫[1,b)]

概要：ロボット産業は 2035 年には 9.7 兆円規模の産業へ成長すると予想されており成長が期待されている．ロボットの増加は，新たなサイバー攻撃のターゲットとされる可能性があり，特に人の近くで動作するようなサービスロボットには深刻な問題になりかねない．様々なハードウェアによる様々な機能を持つロボットの開発にはミドルウェアがよく使用される．本稿では，ミドルウェアの中でも広く使用されている“ロボットオペレーティングシステム（ ROS)”について， 2017 年 12 月にリリースされた次期バージョンの ROS2 に着目する．ROS2 はセキュリティに対応していると言われており，通信の暗号化などがサポートされている．ROS2 の暗号化についてデータ長やサポート OS による特性をテストすることで，ROS2 のセキュリティ機能を実際に適用する際に考慮すべき点について明らかにした．

キーワード：ロボットオペレーティングシステム，ROS，ROS2，ミドルウェア，セキュリティ

Performance evaluation of robot operating system in cryptographic communication

Abstract: The robot industry is expected to grow to an industry of 9.7 trillion yen in 2035, and growth of new fields such as the service field is expected as well as industrial robots. The increase of robots can be targeted by new cyber attacks, and it can be a serious problem especially for service robots that operate near people. Middleware is often used to develop robots with various functions using various hardware. In this paper, we focus on the next version of ROS2 released in December 2017 for “ Robot Operating System (ROS)”, which is widely used in middleware. ROS2 is said to support security, and communication encryption is supported. By testing the data length and characteristics of the supported OS for ROS2 encryption, we clarified the points to consider when actually applying the security function of ROS2.

Keywords: Robot Operating System, ROS, ROS2, Middleware, Security

1. はじめに

近年，様々な分野でロボットの活躍が期待されている．図 **1** は国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO) が 2010 年に公表したロボット産業の将来市場予測 [1] である．ロボット産業は 2035 年には 9.7 兆円規模の産業へ成長すると予想され，中でもサービス分野のような新しい分野が産業用ロボットの規模を超える勢いで成長が見込まれている．

[1] 情報セキュリティ大学院大学
221–0835 神奈川県横浜市神奈川区鶴屋町 2-14-1
[a)] dgs198102@iisec.ac.jp
[b)] okubo@iisec.ac.jp

このようなロボットの増加は新たなサイバー攻撃のターゲットとされる可能性があり，特に人の近くで動作するようなサービスロボットへの攻撃は人体や周りの設備などへの深刻な問題になりかねない．

一方，オープンソースソフトウェアの製品やサービスへの活用は，年々増加しており，個人や大学だけでなく企業にとっても必要不可欠なものとなっている [2] ことから，オープンソースソフトウェアについてセキュリティの問題を検討しておくことは企業にとっても有益であると考える．

ロボットの開発に際しては，様々な機能を持つロボットに対し，より容易に開発を可能にするロボット用のミドルウェアを使うことが主流になってきている．

本研究では，ロボット用ミドルウェアの中でも最も広く

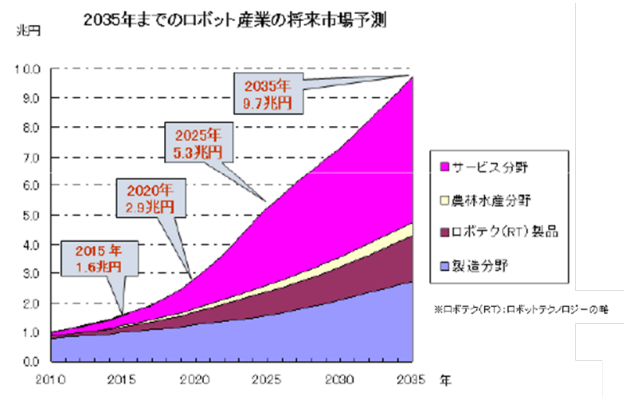

図 1　ロボット産業の将来市場予測：ロボット産業は 2035 年には 9.7 兆円規模の産業へ成長すると予想されている

利用されているオープンソースソフトウェア「ロボットオペレーティングシステム（ROS）」の，特に 2017 年 12 月にリリースされた次期バージョンの ROS2 に着目する．ROS2 はセキュリティ対応がされており通信の暗号化などがサポートされている．この ROS2 暗号化についてリアルタイム性を考慮するため様々な状況で性能測定を行った結果について示す．

2. ROS2 概要

ROS2 とこれまでの ROS(以降 ROS1 と呼ぶ) の違いとしては表 1 のようになっている．ROS1 との互換性は持っておらず，ROS1 と ROS2 の間の通信をサポートするブリッジが用意されている．また，通信方式が大きく変わっており，DDS (Data Distribution Service) を使用する．

表 1　ROS1 と ROS2 の比較
（引用：ROS ロボットプログラミングバイブル）

	ROS1	ROS2
プラットフォーム	Ubuntu, macOS	Ubuntu, macOS, Windows
通信	XMLRPC+ROSTCP	DDS
言語	C++03, Python2	C++14, Python3(3.5+)
ビルドシステム	Rosbuild → catkin	ament
メッセージサービス	*.msg, *.srv	new *.msg, *.srv, *.msg.idl, *.srv.idl
roslaunch	XML	Python
複数ノード対応	1 プロセスに 1 ノード	1 プロセスに複数ノード
グラフ表現	Remapping は起動時のみ	Remapping は実行時も可能。
ネームサービス	マスタ	マスタなし (DDS ミドルウェア)

DDS は OMG で規格化されている通信ミドルウェアであり，複数の実装があるが，特に指定しない場合はデフォルトでは FastRTPS が使われる．

他にはプラットフォームに Windows が含まれていること，DDS の使用に伴い ROS マスタノードが使われなくなっていることが大きな違いである．

また，デフォルトでセキュリティをサポートし，環境変数の設定によりセキュリティ機能の ON/OFF が可能である．サポートされるセキュリティ機能は，通信の暗号化およびノード間のアクセス制御である．

3. 先行研究

ROS2 の先行研究としては，組み込み機器の観点からパフォーマンスを測定したものがある [3]．これは用途からリアルタイム性にフォーカスしており，DDS の複数の実装の比較を行ったものである．ローカル PC 内に閉じたノードだけでなくリモート PC 間や ROS1 と ROS2 の混在の場合についても考慮したテスト条件になっている．DDS の実装により性能が異なることが明らかになり，それらを踏まえて実際にどの DDS 実装を使用するかを検討する必要があると結論づけている．

ただし，セキュリティについては特に触れておらず，また評価している ROS2 がアルファ版となっていることや，現在では実装が中断されている DDS についても検討されている．

また，ROS2 で使用している通信ミドルウェアである DDS 単体に関しては，Pardo らは暗号化に対するオーバヘッドを計測しており [4]，1 %から 40 %のオーバヘッドがあるとの結果を示している．

さらに DDS に限らずパブリッシュ/サブスクライブシステム全体のセキュリティに関しては，Christian らは 1998 年から 2014 年までの学術分野における文献の調査と分析を行なっている [5]．学術文献でのセキュリティ対策と製品レベルのセキュリティ対策にはまだギャップがあるとし，暗号化などのセキュリティ対策はオーバーヘッドになることから，どのようなセキュリティ対策が最良であるかを決定するための適切なセキュリティ評価手段を持つことが大切であるとしている．

さらに，最近では，Belguith らは悪意のあるサブスクライバを無効化する手法を提案している [6]．パブリッシュ/サブスクライブシステムの機能に影響を与えず，新しい鍵の再生成や再配布を行うことなく実現できる手法である．

プライバシーの観点から，Shikfa らはコンテンツベースのパブリッシュ/サブスクライブネットワークにおけるプライバシー問題を取り上げ，暗号化ルーティングテーブルを使うことで，エンドユーザ間で内容を明かさないまま通信のルーティングを行う方法を提案しているものもある [7]．

4. ROS2 暗号通信テスト

4.1 ROS2 想定システム

ROS2 ノードで動作するテスト用の車輪ロボットとコン

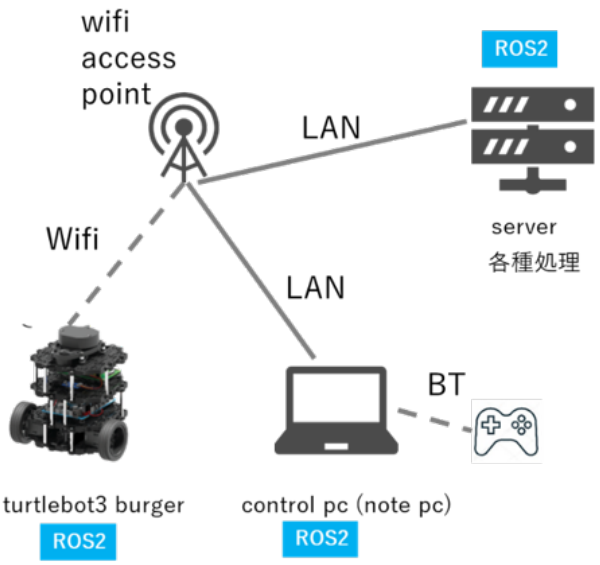

図 2　リモート PC(control pc) と車輪ロボット (turtlebot3) を使った ROS2 想定システム

トロール PC を使うシステムを想定した．構成を図 **2** に示す．

このシステムでは一般的な移動型ロボットが移動に関して行う機能を想定している．ロボットはコントロール PC からの指令により移動を行う．移動中にロボットは自身に搭載された LIDAR からのセンサデータにより周囲の障害物を検知し，周辺地図を作成する．一度地図を作成した後は，コントロール PC から，地図上で目的地を指示することにより自動で目的地まで経路移動を行うことができる．

ノード構成は図 **3** のようになっており，ROS2 ノードは RemotePC 内の 4 つの円で示されている (ノード構成は turtlebot3 emanual より)．想定する悪意者からの攻撃としては，移動指示をなりすまし偽の指示を送信すること，LIDAR のデータを盗聴し他の地図を作成することを想定している．

これまでの先行研究では，ROS2 の平文での通信遅延を計測したものや，DDS 単体での暗号通信測定を行ったものはあるが，ROS2 の暗号通信についてのものはなかった．そこで，本研究ではこの盗聴を想定し ROS2 で暗号通信を行った場合，速度はロボットの動作に影響を与える可能性があるため，暗号通信について通信時間の計測を行った．筆者らはこれまで ROS2 の暗号通信に着目し通信時間を Linux でのローカル PC 内にて測定した [8] が，他の対応 OS について検討はしていなかった．

4.2　ROS2 暗号通信時間測定

ノード間の通信時間に着目するため，ロボットシステムとしてのノード構成ではなく，データをパブリッシュ/サブスクライブするのみのシンプルな機能を持ったノードを用意し，他のノードは実行しない状態でテストを行った．

ROS2 は複数の OS をサポートしていることも特徴となっているため，OS による違いの確認として，Linux，MacOS，Windows を用意しそれぞれの OS 上で ROS2 ノードを実行しテストを実施した．

図 **4** のようにデータを送信するノード 1 とデータを受信するノード 2 を C++にて作成した．

暗号化については ROS2 のデフォルトである TLS を使用し，暗号に使うキーはパブリッシャとサブスクライバのノード毎に作成し PC 内のディレクトリにファイルとして保存した．

4.2.1　送信データ

送信データは図 3 で使用されている LIDAR のデータを送信するためのトピック（/scan）で使われる LaserScan 型を使用し，このデータのサイズを変えて実施した．

LaserScan 型は ROS2 であらかじめ定義されている構造体であり，内容を図 5 に示す．1 回の通信ごとにトピックが受信されるのを確認するために LaserScan 型にある Header 内に 1 通信ごとに通し番号を代入した．また，通信データ量を変更するために，LaserScan 型の ranges 変数の配列サイズ（以降データサイズと呼ぶ）を変更してテストした．データサイズは 100 個から 10^6 個まで 10 倍ごとに変更した．通信時間の測定のためにパブリッシャ側で送信時刻を Header に通信ごとに代入した．それ以外のデータは 0 で初期化した．

4.2.2　テスト環境と構成

テスト環境と接続構成を図 6 に示す．各 OS 違いのノート PC を，有線 LAN でスイッチ（NEC Aterm WG2600HP）に接続した．有線ケーブルも同じ形式の同じ長さのものを使用した．また，PC の接続については，ローカル内のノード同士の通信テスト時はネットワーク線を外し単体でテストを実施し，リモート間のノードテストの場合はテスト対象となる PC のみを接続した．ただし，時刻同期のための NTP サーバを Ubuntu18.04 上で実行したため，テスト対象にこの PC が入っていない場合でも，この PC はネットワークに接続した．また，各ノート PC の無線 LAN 側の設定は無効に設定することで，不要な無線 LAN 通信が発生しないようにした．

実際に使用した PC と ROS2 のスペックを表 **2** に示す．各 OS が動作する PC においてスペックの違いに大きな差がないように配慮した．また，スペック違いの比較のため，OS が同じ（Ubuntu18.04）であるがスペックが違う PC もテスト対象とした．

ROS2 のバージョンについてはすべて crystal clemmys (2018 年 12 月リリース) を使用し，公式手順[*1] にしたがってバイナリパッケージとしてインストールした．DDS についてもすべての PC において ROS2 デフォルトの FastRTPS

[*1] https://index.ros.org/doc/ros2/Installation/Crystal/

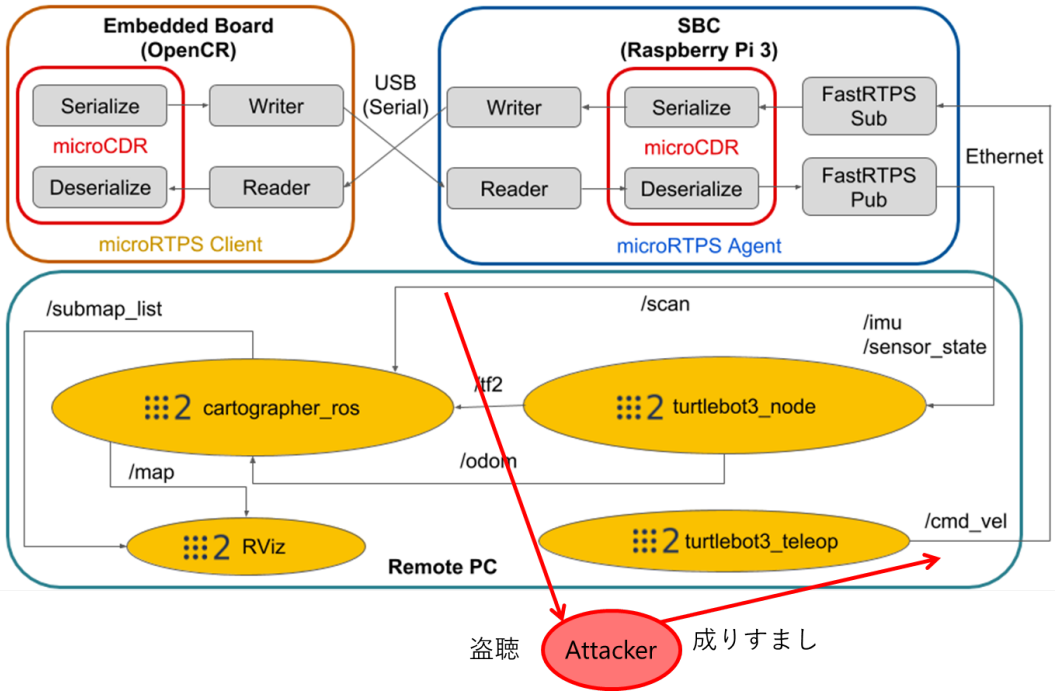

図 3　想定システムでのノード構成と悪意者による脅威 (RemotePC 内の 4 つの円が ROS2 ノード)(ノード構成引用 turtlebot3 emanual)

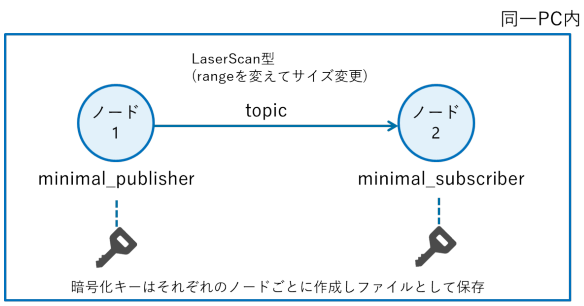

図 4　ROS2 における通信テストノード

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

std_msgs/Header header # timestamp in the header is the acquisition time of
                      # the first ray in the scan.
                      #
                      # in frame frame_id, angles are measured around
                      # the positive Z axis (counterclockwise, if Z is up)
                      # with zero angle being forward along the x axis

float32 angle_min     # start angle of the scan [rad]
float32 angle_max     # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment # time between measurements [seconds] - if your scanner
                       # is moving, this will be used in interpolating position
                       # of 3d points
float32 scan_time      # time between scans [seconds]

float32 range_min      # minimum range value [m]
float32 range_max      # maximum range value [m]

float32[] ranges       # range data [m]
                       # (Note: values < range_min or > range_max should be discarded)
float32[] intensities  # intensity data [device-specific units].  If your
                       # device does not provide intensities, please leave
                       # the array empty.
```


図 5　テストに使用した LaserScan 型

を使用した．

通信テスト用のプログラムは ROS2 example[*2] にある minimal_publisher，minimal_subscriber の C++版を修正して使用し，各 OS にてコンパイルして作成した．

[*2] https://github.com/ros2/examples

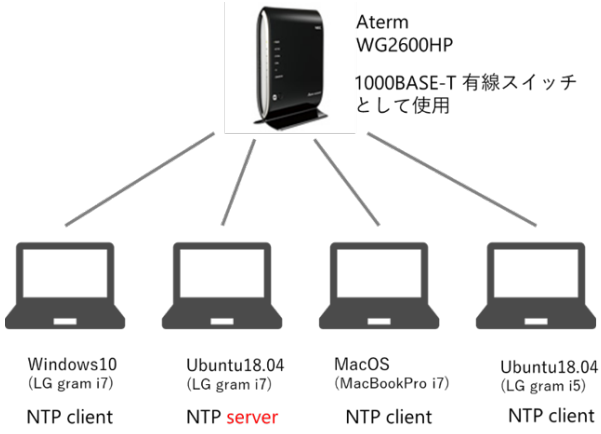

図 6　テスト時の構成と接続図：各 PC を有線 LAN でスイッチに接続し，Ubuntu 上にて NTP サーバを動作させた

表 2　テストに使用した複数 OS の PC スペック: ROS2 がサポートしている OS が動作している PC を使用

	Linux5	MacOS	Windows10	Linux7
OS	ubuntu 18.04LTS	macOS 10.14.4	Win10 pro	ubuntu 18.04LTS
CPU	i5 1.6GHz	i7 2.7GHz	i7 1.8GHz	i7 1.8GHz
ROS2	crystal	crystal	crystal	crystal
DDS	FastRTPS	FastRTPS	FastRTPS	FastRTPS

暗号通信のためのキーは ROS2 のデフォルトコマンドを使用して Ubuntu18.04 上でノード毎に作成したものを，各 PC にコピーして共通のキーとして使用した．

4.2.3 テスト方法

ノード 1 は 100ms ごとに LaserScan 型のデータをトピックとして送信する．ノード 1 はパブリッシュ時の時刻を送信時刻として送信データの Header に記録し送信する，ノード 2 がデータを受信した際の callback が呼ばれた時刻を記録し，受信したデータに代入されている送信時刻との差分を取ることで通信時間とする．100 回の送信分の平均を計算し結果とする．暗号化については標準手法である環境変数による設定を行い，この環境変数の ON/OFF で暗号の ON/OFF を実施した．また，リモート間の場合は時刻が合っていないと差分による通信時間が正しく測定できないため，各テストの前に NTP コマンドにて時刻の確認を行ってから送信を開始する．手順を以下に示す．

(1) 平文テストか暗号テストかによって環境変数を設定する（ノード 1 側，ノード 2 側とも）
(2) ntp コマンドを使用し ntp サーバとの時刻ずれを確認する
(3) ずれが 1ms 以上の場合は，ntpdate コマンド（または ntpclient ソフト）を使い STEP モードで強制的に時刻同期を行う
(4) 1ms 以下にならない場合は何度か時刻同期コマンドを繰り返す
(5) 決められたデータサイズ用のサブスクライバ（ノード 2）を起動する
(6) 決められたデータサイズ用のパブリッシャ（ノード 1）を起動する
(7) 通信回数を通し番号で確認し，100 回を超えたことを確認してノード 1 を停止する
(8) ノード 2 を停止する
(9) データがファイルに保存されていることを確認する
(10) 平文/暗号，およびデータサイズを変更し上記を繰り返す

5. テスト結果

5.1 同一 PC 内での通信時間測定結果

それぞれの PC 内で 2 ノード間のトピック通信を行い，データサイズと暗号設定を変えた場合の測定結果を図 7 に示す．

5.2 リモート PC 内での通信時間測定結果

Linux7-Linux5 間のリモート通信測定結果を図 8 に示し，Linux7-Windows 間の測定結果を図 9 に示す．ここで Linux → Windows の測定結果においてデータサイズが 10^6 個の時，サブスクライバ側にデータが届いたのは暗号化したときで 12 回，平文の時で 3 回であった．

また，Linux7-MacOS 間のリモート通信測定結果を図 10 に示す．

6. 考察

6.1 ローカル PC 内での ROS2 通信について

図 7 によると，平文の場合，通信時間は Linux，MacOS ともにデータサイズが 10^4 個程度までは 0.5ms 程度であるが，それを超えると 1ms 以上まで増加している．さらにデータサイズが増え 10^6 個になると MacOS では通信時間が 4ms 程度であるが，Linux では 10ms 程度必要となる．Windows の場合は他の OS に比較すると全体的に通信時間が必要となり，データサイズが 10^3 個程度までは 0.9ms 程度，10^4 個の時点で 1.8ms 程度と他の OS に比べ 3 倍の時間が必要となっている．

また，暗号化した場合，Linux と MacOS について通信時間はデータサイズが 10^4 個程度までの比較的小さいうちは，平文の約 1.6 倍の 1ms 程度であるが，データサイズが 10^5 個以上に増えてくると平文の時に比較して 2 倍程度通信時間が必要となり，Linux では最大 20ms 必要となっている．

一方で Windows の場合はデータサイズが 10^4 個程度までは，データサイズが増加しても平文の場合と比べ 1 倍から 1.4 倍程度とほぼ同程度で推移するが，10^4 個を超えると暗号化した方が通信時間が短いという逆転現象が生じている．データサイズが 10^6 個の場合は暗号化したほうが平文の時の半分近い 30ms 程度での通信時間となっている．Wireshark でパケットを確認したが，暗号化した場合のデータが特段少なくなっているという現象は確認できなかった．Linux や MacOS の場合は，通信に必要なリソースよりも暗号/復号に必要なリソースが通信性能に対して支配的であるのに対し，Windows の場合は暗号/復号よりも通信にリソースを多く必要としているのではないかと考えられる．

リアルタイム性が必要とされるロボットの場合，単純にシステム内のノード間通信を暗号化することは，通信時間の増加が動作に影響を与える場合があると考えられる．実際のロボットシステムではノード数は今回実験した 2 ノードより多く存在することが一般的であり，その場合はさらに通信時間が増加すると考えられる．ただし，この時間の遅れが具体的にリアルタイム性にどう影響するのかまでは本研究では試せていない．

これらから，データサイズが 10^5 個を超えるような場合は特に通信時間をよく検討した上で暗号化通信を使用することが重要であると言える．また，ノードの増加につれてノード間の通信も増えていくと考えられることから，セキュリティのためにすべてを暗号化するのではなく，それぞれのロボットシステムに合わせて性能とセキュリティリスクから適切に選択することが大切といえる．

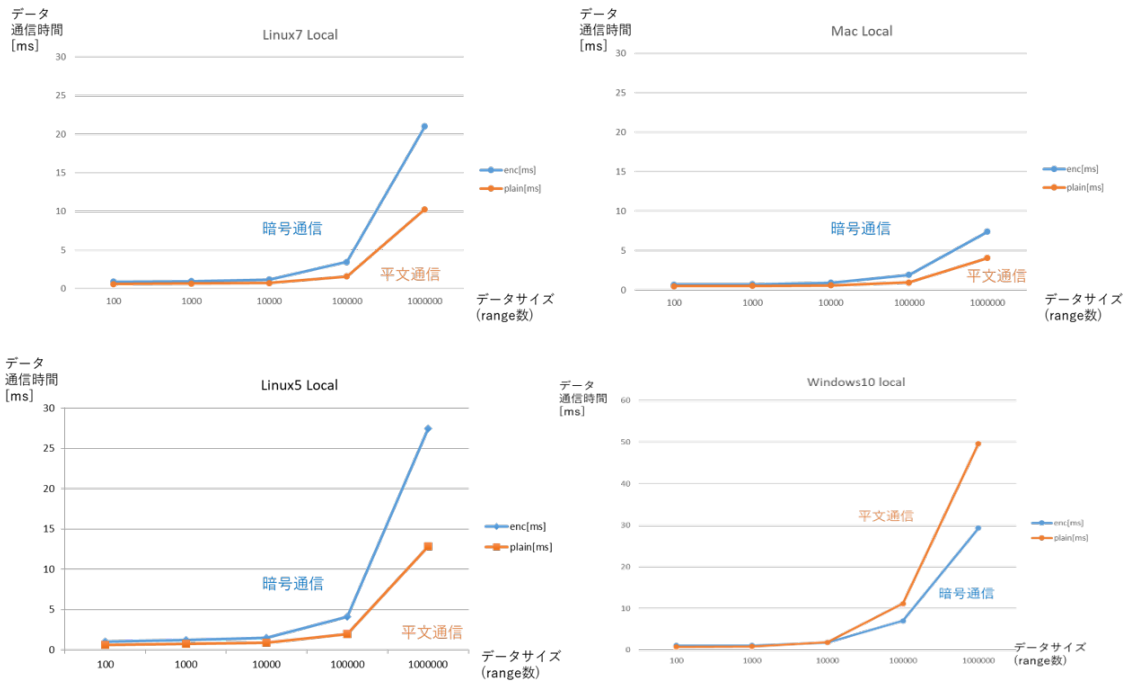

図 7　ROS2 における同一 PC 内 2 ノード間の平文および暗号通信時間測定結果
(左上 Linux7, 右上 MacOS, 左下 Linux5, 右下 Windows)

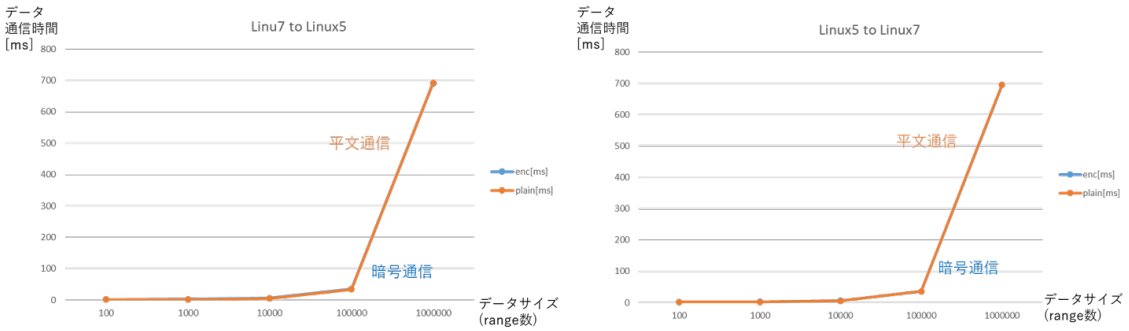

図 8　Linxu7-Linux5 間のリモート通信測定結果
(左 Linux7 → Linux5, 右 Linux5 → Linux7)

6.2　リモート PC 間での ROS2 通信について

Linux 間のリモート通信結果（図 8）をみると，Linux7 と Linux5 についてはどちらの方向の通信についても同じような傾向を示している．データサイズが 10^4 個程度までは暗号化した場合のほうが平文に比べると 1.2〜1.6 倍の時間が必要となっている．

また，データサイズが 10^6 個以降は平文の場合でも暗号の場合でもそもそも 100ms では通信できず，1 通信に 700ms 程度必要となる．通信テストの際には，パブリッシャは 100ms で無制限にデータを送信することはせず，サブスクライバが 1 通信のデータを受け取るたびに，それを待ってから次の送信をするような動作が見られた．結果として途中の通信が抜けてしまうことはなかった．このメカニズムについては本研究では検討していないが，DDS の QoS の設定が関係していると思われる．

次に，Linux-Windows 間の通信について，図 9 を見ると，Windows → Linux の通信（A）と，Linux → Windows の通信（B）で傾向が違っていることがわかる．A の場合は，データサイズの増加につれて通信時間が増えていくが，10^4 個を超えたあたりから暗号文より平文の方が通信時間がかかっている．この傾向は Windows のローカル PC 内での通信結果と同様の傾向である．また 10^6 個になると通信時間は 760ms 程度必要となり，通信は 100ms 周期では回らなくなるが，この場合も Linux 同士のリモート通信の場合と同様に，パブリッシャが通信周期をサブスクライバに合わせることで，100 回の通信に抜けが生じることはなかった．B の場合は，データサイズが 10^4 個までは平文の場合は 10ms 以下で通信ができているが，暗号文の場合はデータサイズが 10^3 個で 27ms，10^4 個で 100ms と急激に通信時間が増加している．データサイズが 10^5 個では平文も暗号文もどちらも 20s 程度もかかっており，この状態では実用にはならないと考える．また，データサイズが 10^6

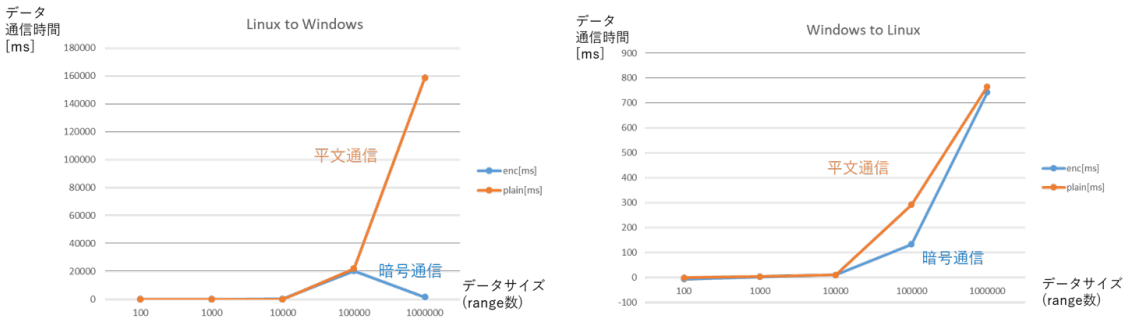

図 9　Linux7-Windows 間のリモート通信測定結果
(左 Linux7 → Windows, 右 Windows → Linux7)

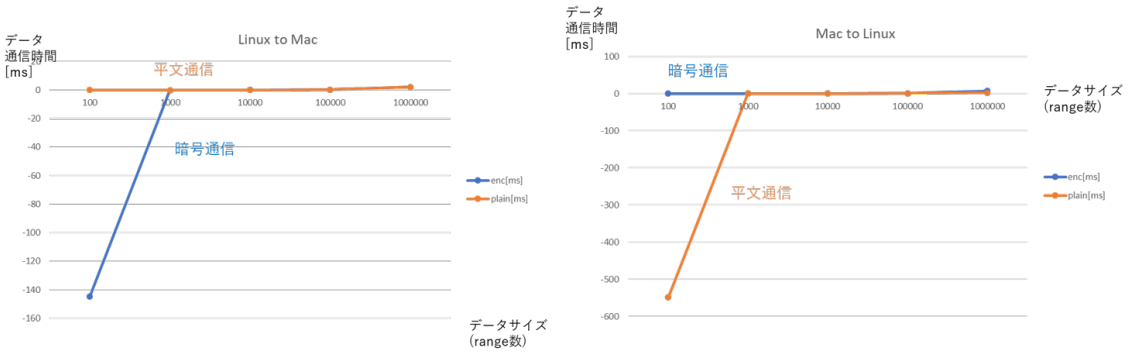

図 10　Linxu7-MacOS 間のリモート通信測定結果
(左 Linux7 → MacOS, 右 MacOS → Linux7)

個の場合は，平文では 158s，暗号文では 1.7s と大きく開きが生じているが，100 回送信のうち実際に通信ができた回数は平文で 3 回，暗号文で 12 回であり，通信時間の問題よりも用途によっては通信自体が成り立たない可能性がある．

A と B を比較すると Linux 側がパブリッシャの場合はサブスクライバの受信を待ってから送信することができるが，Windows 側がパブリッシャの場合はその機能がうまく動作していないためにこのような差になっていると思われる．もし，Windows と Linux 間で ROS2 の通信を行うような場合は，通信方向による特性の違いについても理解しておく必要がある．

また，Linux-Mac 間の結果（図 10）を見ると，データサイズが小さいときに通信時間が -140ms などとマイナスの値になってしまっている．これはおそらくリモート PC 間での時刻同期の問題が原因と考えられる．データサイズが小さいときは，通信時間も短く時刻のずれの影響が大きく結果に表れたと考える．

ローカル内での通信結果，およびリモート間での通信結果を総合すると，ROS2 は Linux，MacOS，Windows に正式対応しているが，実際には OS の違いによって動作が異なることがわかった．特に Windows を使用する際には，データサイズの増加により，急激にパフォーマンスが低下することもあり，システムアーキテクチャを検討する際は，データサイズだけではなく OS についても，検討しておく必要があるといえる．

7. おわりに

ロボットの開発に広く使われている ROS について，特に最近リリースされた ROS2 に注目し概要を調査した．セキュリティ機能については ROS2 ではノード間の暗号化とアクセス制御がサポートされているが，特に暗号化についてはロボットシステムでは通信時間が動作に影響を与える場合もあり，暗号化と平文での通信時間について計測した．ROS2 が対応している Linux，MacOS，Windows の各 OS 上の ROS2 を用いて通信テストを行うことにより，データサイズの増加に対する傾向と合わせて OS の違いによる傾向の違いを明らかにした．今後の課題としては，暗号化に必要なキーの管理方法についての検討，およびもう一つの ROS2 のセキュリティ機能であるノード間の権限設定についての課題調査があげられる．

参考文献

[1] NEDO: ロボットの将来市場予測を公表，https://www.nedo.go.jp/news/press/AA5_0095A.html.
[2] 野村佳秀，木村功作，福寄雅洋，谷田英生：『オープンソースソフトウェア工学』シリーズ企業における OSS 活用の実際 (2016).
[3] Maruyama, Y., Kato, S. and Azumi, T.: Exploring the Performance of ROS2, *Proceedings of the 13th Interna-*

tional Conference on Embedded Software, EMSOFT '16, New York, NY, USA, ACM, pp. 5:1–5:10 (online), DOI: 10.1145/2968478.2968502 (2016).

[4] Gerardo Pardo, R. W.: DDS Security concepts for SROS, `https://ruffsl.github.io/IROS2018\_SROS2\_Tutorial/content/slides/Background.pdf`. (Accessed on 04/14/2019).

[5] Esposito, C. and Ciampi, M.: On Security in Publish/-Subscribe Services: A Survey, Vol. 17, No. 2, pp. 966–997.

[6] Belguith, S., Cui, S., Asghar, M. R. and Russello, G.: Secure Publish and Subscribe Systems with Efficient Revocation, *Proceedings of the 33rd Annual ACM Symposium on Applied Computing - SAC '18*, ACM Press, pp. 388–394.

[7] Shikfa, A., Önen, M. and Molva, R.: Privacy-Preserving Content-Based Publish/Subscribe Networks, *Emerging Challenges for Security, Privacy and Trust* (Gritzalis, D. and Lopez, J., eds.), Vol. 297, Springer Berlin Heidelberg, pp. 270–282.

[8] 亘理克好，大久保隆夫：ロボットオペレーティングシステムのセキュリティ機能に関する考察，情報処理学会研究報告 (2019).