特別研究報告

題目

IoT プラットフォーム連携に基づく
実世界センシングによる
実環境の異常検知システムの実装と評価

指導教員

村田 正幸 教授

報告者

山田　諒真

2021 年 2 月 9 日

大阪大学 基礎工学部 情報科学科

令和 2 年度 特別研究報告

IoT プラットフォーム連携に基づく実世界センシングによる実環境の異常検知システムの実装と評価

山田　諒真

内容梗概

モバイルデバイスの普及などにともない、現実世界の状況をセンシングし分析処理を行うことにより、新たなサービスを提供する実世界センシングが注目されている。しかし、実世界センシングを伴う IoT（Internet of Things）サービスの種類が多くなると複数のサービスに対して同一のセンシングデータを送信する通信の重複が発生し、通信の負荷や遅延の増大が予想される。この問題への対処として、サービスごとに独自に情報を集めるのではなく、複数のサービスに対するセンシング情報を集約する IoT プラットフォームを構築し、IoT プラットフォームの情報連携によってサービスを実現する方法が考えられる。

そこで本報告では、IoT プラットフォーム間の情報連携を可能とする IoT プラットフォームを実機に実装し、様々なセンシングデータ量とセンシング周期において発生する遅延を測定した結果にもとづいて、IoT プラットフォーム連携により実現可能なユースケースを整理する。遅延測定に用いるサービスシステムとして、建物に設置したカメラの画像を蓄積する IoT プラットフォームと交通管理を行う IoT プラットフォームが連携し、建物周辺の画像を縮約したデータの送受信によって道路上の障害物情報を検知する異常検知システムを FIWARE を利用して実装した。実装したシステムでは、画像の RAW データから深層学習における中間層の出力値をエンティティとし、Pub/Sub 通信によって送受信している。IoT プラットフォームの設置場所としてエッジ／クラウドを想定し、エッジ連携シナリオ、エッジ／クラウド連携シナリオ、クラウド連携シナリオの 3 種の情報連携シナリオにおいて、画像枚数および送信を行うエンティティの個数を変えつつ、Pub/Sub 通信に要する時間と画像情報処理に要する時間の和である連携遅延を測定した。測定の結果、エッジ連携シナリオにおける連携遅延が約 470 ms となった。クラウドを利用するエッジ／クラウド連携シナリオ、クラウド連携シナリオでは、それぞれの連携遅延は約 475 ms、約 1000 ms となり、Pub/Sub 通信に要する時間の増大が情報量の増大とともに顕在化することが明らかとなった。さらに、測定結果にもとづいて、エッジ連携、クラウド連携により実現される IoT サービスの適用領域について、時間粒度と空間粒度の観点から整理した。

主な用語

実世界センシング、IoT（Internet of Things）、IoT プラットフォーム、プラットフォーム連携、FIWARE (Future Internet WARE)、Pub/Sub メッセージング、エッジ連携、クラウド連携

目 次

図 目 次

表 目 次

1　はじめに

センサーデバイスの小型化や低価格化と、モバイルデバイスの普及を背景に、現実世界の状況をセンシングし分析処理をおこなうことにより、新たなサービスを提供する実世界センシング技術が注目されている。例えば現在、カメラの技術進展により量産品でも高い画素の画像や映像が大量に収集できるようになっている。そして、この画像・映像を用いて分析処理を行うことによって、これまでのカメラでは実現不可能であった数百人〜千人規模の同時監視が可能になり、スタジアムなどの防犯・管理などのサービスを実現することも考えられている [1]。さらに将来は、特定のサービスやアプリケーションに特化した実世界センシングではなく、例えばスタジアムの動きと連動して周辺道路の交通流を制御することなど、収集した情報を異なるサービスで共有しつつ、より優れたサービス・アプリケーションの提供も考えられつつある [2]。

このような IoT サービスを実現するためには、まず現実世界から大量のセンサデータを収集する必要がある。そのため、大量のセンサデータを収集すると、複数の種類のデータが集まるため、それぞれのデータに適した分析・管理方法についても考える必要がある [3]。また、IoT サービスにおける情報処理は、個々のサービスが個別に実施するものと考えられている。従って、多数のセンサを用いて多数のセンシングサービスを利用しようとすると、複数の異なる IoT サービスに対して同一のセンシングデータを送信する通信の重複が発生する。その結果、情報収集のための通信量が増大してネットワーク負荷が高まり、サービス遅延が増大することが予想される。この問題への対処として、サービスごとに独自に情報を集めるのではなく、複数のサービスに対するセンシング情報を集約する IoT プラットフォーム [4] を構築し、IoT プラットフォームの情報連携によってサービスを実現する方法が考えられる。

本報告では複数のサービスに対するセンシング情報を実世界上の拠点に集約する IoT プラットフォームを構築し、プラットフォーム間の情報連携によって IoT サービスを構成することに着目する。 IoT プラットフォームをセンサデータの発生源となる実世界上の組織等に設置し、これまで個々のサービスが個別に提供していたセンシングデータ収集やセンシングサービスにおける情報処理を集約することを考える。これにより、現在の IoT プラットフォームで懸念される通信負荷と情報処理負荷の軽減が期待できる。ただし、IoT サービスを構成するためには、あるプラットフォームで集約している情報を他のプラットフォームでも利用することが必須であり、プラットフォームが集約しているセンシング情報を共有するための情報交換プロトコルが必要となる。また、IoT プラットフォームが実現するサービスには、人の動きや車両の動きなどの数秒単位で状態が変わる情報を扱うものから、街の建造物の情報などの数日・数年経過しても変化がほとんど起こらない情報を扱うものまでの、様々な時間粒度のサービスが混在することが予想される。

そこで本報告では、IoT プラットフォーム間の情報連携を可能とする IoT プラットフォームを実機に実装し、様々なセンシングデータ量とセンシング周期、さらに複数の情報連携シナリオにおいて発生する遅延を測定し、IoT プラットフォーム連携により実現可能なユースケースを整理する。IoT

プラットフォームをサービス毎ではなく組織毎に構築することで、センシングデータの所有組織自身がデータ管理可能となる。遅延測定に用いるユースケースとして、建物に設置したカメラの画像を蓄積する IoT プラットフォームと交通管理を行う IoT プラットフォームが連携し、建物周辺の画像を縮約したデータの送受信によって道路上の障害物情報を検知する異常検知システムを想定し、そのプロトタイプ実装を行った。異常検知システムでは、道路上の障害物情報は道路を所管する組織が把握し車両へと配信することが基本となる。IoT プラットフォーム連携では、道路上の障害物を把握するにあたって別の組織の建物等周辺を観察する定点カメラの情報を取得し、1 つの組織のみでは把握できない道路の異常を速やかに検知し、事故や混雑防止に繋げることが期待される。実装したシステムにおいては、画像の RAW データから深層学習における中間層の出力値をエンティティとし、Pub/Sub 通信によって障害物情報に相当する情報を送受信するシステムとなっている。なお、IoT プラットフォームは、サービスによってエッジに配置する場合とクラウドに配置する場合が考えられる [5]。そこで、IoT プラットフォームをエッジに配置し連携を図るエッジ連携シナリオと、IoT プラットフォームをクラウドに配置し連携を図るエッジ／クラウド連携シナリオ、クラウド連携シナリオの 3 種の情報連携シナリオにおいて、Pub/Sub 通信に要する時間と画像情報処理に要する時間の和である連携遅延を測定する。時間粒度と空間粒度の異なる IoT プラットフォーム連携のユースケースを示すとともに、測定した連携遅延に基づくエッジ連携、クラウド連携の適用領域を明らかにする。

本報告の構成は以下の通りである。まず 2 章では、現在の IoT サービスについて行われている研究や、本報告における IoT プラットフォーム連携について述べる。次に 3 章では、異常検知システムにおける、IoT プラットフォームのクラウド連携、エッジ連携それぞれにおける連携遅延を測定することにより、それぞれの性能評価を行い、その方法、結果を述べる。そして 4 章では、3 章の結果を用いて、クラウド連携、エッジ連携それぞれにおけるユースケース適用領域について述べる。最後に 5 章では、本報告のまとめと今後の課題を述べる。

2 IoT プラットフォーム

2.1 IoT

IoT (Internet of Things) とは、インターネットに接続されていなかったものが、ネットワークを通じてサーバーやクラウドと接続して相互に情報交換をする仕組みである。これにより、今まで計測することができなかったデータなどを計測することができ、新たなサービスを実現することができている。そのため現在では、IoT を利用したサービスの研究が盛んにおこなわれている。

文献 [6] では、スマートホーム実現のために IoT を利用し、そのトラフィックを分析することにより、IoT を利用して実現したスマートホームが、家庭内でプライベートなアクティビティを実現できていることを示している。スマートホームのための IoT デバイスは近年急増しており、その多種化によって、家庭内のプライバシー保護の実現が新たな課題となっていた。そのためこの研究では、プライバシー保護のための新たな防御策の開発・研究を行っている。

文献 [7] では、農業で IoT を利用し、目的の土壌水分量を監視および維持を目的とした土壌水分センサシステムに IoT を利用している。このシステムでは、農民が土地耕作の際に汲み上げる水の浪費や不足を防ぐために、センサによって現在の水分量を計測し、目的の土壌水分量より現在の水分量が増減している場合に警告メッセージを作成し、農民に知らせる。

文献 [8] では、街の水管理システムで IoT を利用し、水消費量の異常検出を行うために IoT を利用している。スマートメーターなどを利用して家庭内の水消費量を計測し分析を行う。そして、家庭内の水消費量が多く水の浪費の恐れがあると判断できた場合、消費者に警告メッセージを作成し、消費者に知らせる。これによって、消費者の水の浪費を防ぐことができる。

このように IoT は、家庭内や農業など様々な分野でサービスに利用されている。これらに対し本論文では、IoT を利用する分野として、交通・防犯の分野に着目し、IoT を利用し道路上の異常検知を行うサービスシナリオの実装と評価を行っている。

2.2 垂直統合型の IoT プラットフォーム

IoT サービスをユーザに提供するために利用される IoT プラットフォームの種類の 1 つとして、垂直統合型の IoT プラットフォームがある。垂直統合型の IoT プラットフォームは、1 つの IoT サービスに対して、サービスの実現に必要な機能やモジュールをすべて 1 つのプラットフォームに実装し、サービスを実現するプラットフォームである。この構造のプラットフォームは、1 つの IoT サービスに特化しているため、機能性が高いという長所がある [9]。

この垂直統合型プラットフォームを利用したサービスとして、富士通の垂直統合型データシェアハウス基盤 [10] がある。

このサービスでは、膨大なリアルタイムデータ活用に向けて分析やアクションのスピードを上げるために、1 つの垂直統合 IoT プラットフォームですべての技術を実現している。このサービスで

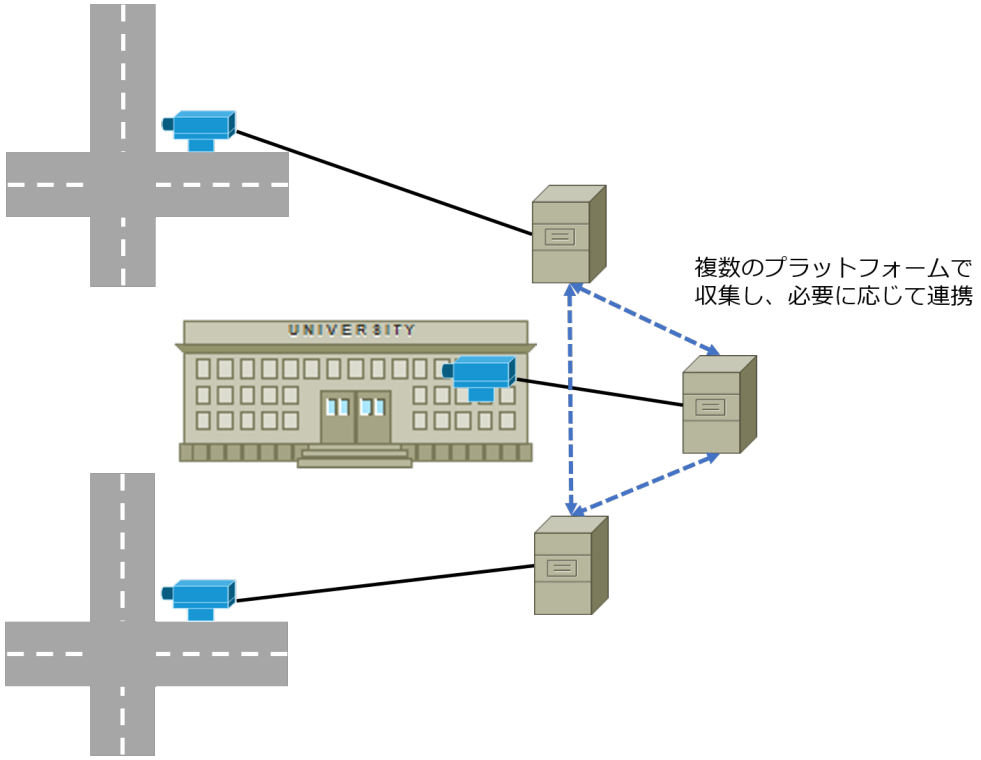

図 1: IoT プラットフォーム連携

実現している技術には、大量データの高速収集・解析技術、センサデータの高速マッピング技術、非構造データからの新たな情報生成技術がある。

2.3 IoT プラットフォーム連携

前節で述べたような 1 つの IoT プラットフォームですべての処理を行う場合、大量かつ複数の種類のあるセンサデータをそれぞれの種類に適した分析・管理を行う必要があるため、通信負荷や情報処理負荷が増大することが懸念されている。また垂直統合型のプラットフォームは、1 つの業種に特化した構造になっているため、汎用性に欠けている。

このような課題を解決するために、複数の IoT プラットフォームを利用し、必要に応じてプラットフォーム同士が連携することを考える。これにより、それぞれの IoT プラットフォームで 1 つの種類の情報を収集することができるようになり、これまで、1 つの IoT プラットフォームにおいてすべての処理を行っていたために発生していた通信負荷や情報処理負荷を軽減することができる。また、連携によって組み合わせる情報を工夫することにより、汎用性のあるサービスを実現することも可能となる。IoT プラットフォーム連携について図 1 に示す。

図 2 のように、センサデータを複数のプラットフォームで分けて収集することにより、1 つのプ

ラットフォームの通信負荷や情報処理負荷を軽減することができる。

しかし、プラットフォーム連携を行う場合、あるプラットフォームで集約している情報を他のプラットフォームでも利用しつつサービスを構成することが必要となり、情報利用に要する通信負荷や情報処理負荷、通信遅延は依然として生じる。また、連携を行うための情報交換プロトコルなどの実装が必要となるため、1 つのプラットフォームで実現できるサービスよりも複雑な構造となる。

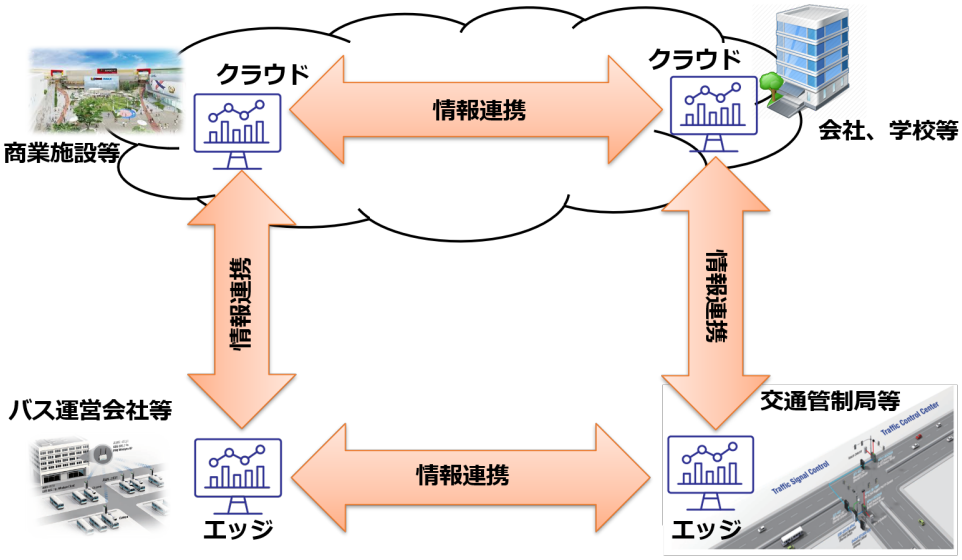

図 2: 想定するシステム

3　プラットフォーム連携に基づく異常検知システムの実装と評価

3.1　想定するシステム

想定しているシステムは、交通・防犯を組み合わせたシステムである。

このシステムは、それぞれの建物 (商業施設、会社、学校等) から得られる情報と、バス運営会社や交通管制局等から得られる情報を組み合わせることによって、現在のそれぞれの建物や道路の情報の管理、人や障害物との衝突などの事故回避、不審者からの防犯などを目的としたシステムである。

このシステムのイメージ図を図 2 に示す。

このシステムは、それぞれの建物の情報を集約するサーバと、その周辺やそれ以外の道路情報を管理するバス運営会社や交通管制局等が所管するサーバと、それらのサーバに画像データを送信するカメラから構成されている。そして、必要に応じてサーバ同士が連携することによって、衝突などの事故回避、現在の道路情報の伝達、現在の建物内外情報の伝達、不審者からの防犯を行うことを目的としている。また、サーバ同士が連携し情報を交換するとき、場合によっては接続されているカメラで得た画像そのものではなく、連携に必要な画像の縮約情報を共有する。

なお、建物のサーバは、エッジではなくクラウドに配置することも考えられる。当然、クラウドに配置することによる NW 遅延や情報量の違いが生じるため、クラウド型プラットフォームを利用した場合の通信量やデータサイズ、発生する遅延を計測することで適用可能なユースケースを整理する必要がある。

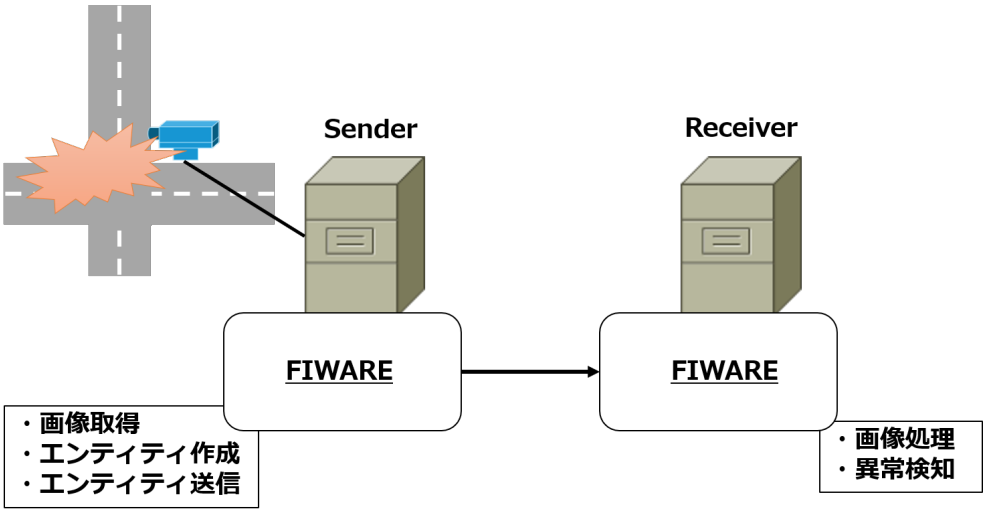

図 3: 異常検知システムのプロトタイプ実装

3.2 実装したシステム

実装したシステムは、道路上の障害物情報 (道路への落下物、道路上の倒木、道路上の違法駐車車両等) の異常検知を行うシステムのプロトタイプである。実装したシステムでは、道路上の障害物情報などの数分、数時間以上経過しても状況の変化が起こらない情報に着目し、これらの情報のみを扱うシステムのプロトタイプである。また、実装したプロトタイプでは、1 本の道路の障害物情報に着目するため、狭範囲から取得した情報の処理を行うシステムとした。さらに、実装したプロトタイプでは、道路上の障害物情報を認識することができる画像のみを扱うシステムとした。

まず、画像の RAW データから深層学習における中間層の出力値をエンティティとする。そして、あらかじめ実装した他の IoT プラットフォームとのデータ共有を行う機能モジュールでの情報共有を行う。そして受信側で出力値を用いて処理を行い異常検知を行う。

プロトタイプの実装には、低コストかつ効率的な基盤開発が可能で、多様なデータの統合管理を実現可能な点から、IoT プラットフォームのオープンソース実装である FIWARE (Future Internet WARE) を用いる。実装したシステムの概要を図 3 に示す。

3.2.1 FIWARE

FIWARE は 7 つのカテゴリ、約 40 種類のモジュール群で構成されている。そして、用途に合わせてこれらのモジュール群を自由に組み合わせて利用する。FIWARE の各モジュールは、OMA（Open Mobile Alliance）で標準化された NGSI（Next Generation Service Interface）と呼ばれる異なるアプリケーション間でデータ連携をするためのインタフェースで規定されており、これを通してデータの受け渡しを行う。

FIWARE のモジュールはいずれも、OSS (Open Source Software) をベースに開発されている。そして、そのリファレンス実装も Web で広く公開されているため、その雛形を参考に、必要なモジュールを組み合わせたり、機能を独自に追加したりなどの作業で、低コストかつ効率的な基盤開発が可能である。また、FIWARE では、多様なデータをコンテキストデータとして格納する。また、ネットワーク経由でのデータ検出／取り込みのための API も用意されている。これによって、多様なデータの統合管理を実現することができる。

セキュリティ面において、FIWARE では、既に認証／認可の実現のため、ID 管理を行うモジュールである Keyrock [11] と、管理において許可されたユーザのみ Orion Context Broker へのアクセスを許可するモジュールである Wilma [12] の 2 つのセキュリティに関するモジュールが実装されている。

しかし、現在 FIWARE のセキュリティとして実装されているのは、前述のような Orion Context Broker へのアクセス制御のみである。そのため、認証が通った先のセキュリティは存在していない。例えば、連携において送受信されるそれぞれのデータについて、ある組織のみが解読できるような手法を施したり、改ざん防止のための手法を施すことができない。そのため、今回の研究で実装を行うプラットフォーム連携において、セキュリティ対策がされたプラットフォーム連携を実現するためには、送受信に用いられるそれぞれのデータに情報を保護するための手法を実装することが必要がある。

3.2.2 プラットフォーム連携

実装するシステムでは、FIWARE のコンテキストデータ送受信に関する機能を実現しているモジュールである Orion Context Broker を利用した、「Context Broker フェデレーション」と呼ばれる方法を利用して、Context Broker 間の Pub/Sub メッセージングによる、プラットフォーム連携を実装している。なお今回の実装での Publisher は送信側、Subscriber は受信側、Topic は画像情報としている。

この方法の概要を図 4 に示す。

この方法では、まず、あらかじめ送信側の Context Broker に、送信側の Context Broker で情報の更新がなされたとき、更新したエンティティを受信側に送信するようにサブスクリプションの作成を行う。

このサブスクリプションの設定を行うと、送信側で新たにエンティティを作成したときに、Context Broker が、事前に定義したサブスクリプションを確認し、サブスクリプションに定義されている属性と新たに作成したエンティティの属性が一致しているか確認する。そして属性が一致した場合、サブスクリプションに定義されている受信側の Context Broker を確認し、その Context Broker に向けて新たに作成したエンティティを送信する [13]。

また、今回の実装で使用した Context Broker フェデレーション以外のプラットフォーム連携の

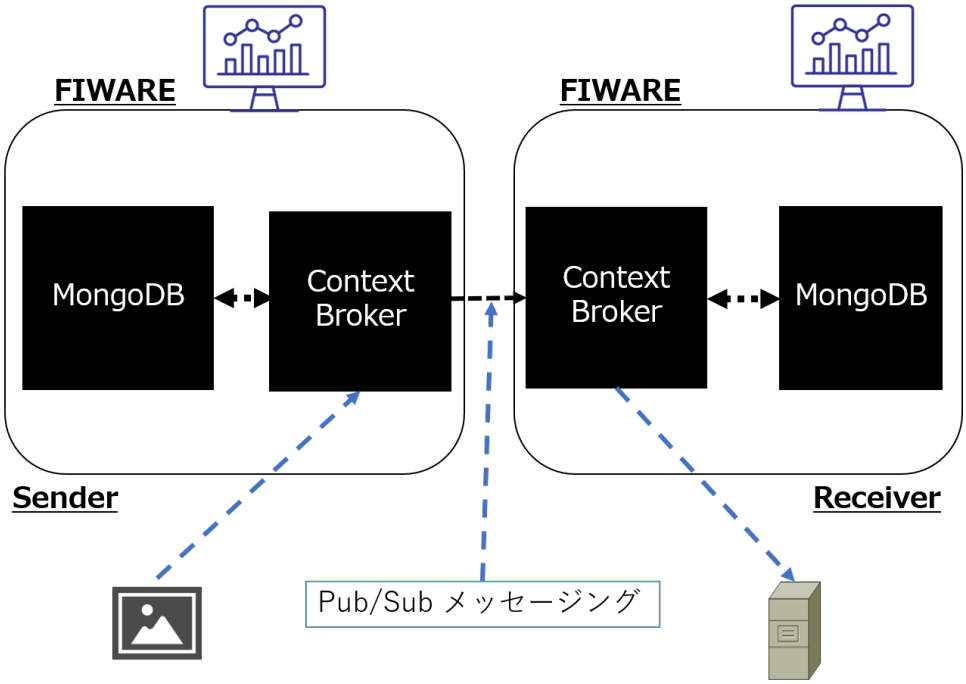

図 4: Context Broker フェデレーション

方法として、リアルタイムのパブリッシャ・サブスクライバ間の通信を提供する RTPS (Real Time Publish Subscribe) プロトコルを利用する連携がある [14]。

しかし、このプラットフォーム連携を利用するためには、新たにモジュールを実装する必要があり、かつ、実装するシステムでは、リアルタイムでの通信は考えていない。そのため今回の実装では、Context Broker を利用してプラットフォーム連携を実現できる Context Broker フェデレーションを利用している。

3.2.3 画像処理

実装したシステムでは、Residual Net (Res Net) [15] を送信側受信側両方で動作させ画像の処理を行っている。送信側の Res Net では、画像の RAW データから中間層の値を出力値として取り出し、受信側の Res Net では送信側の出力値から特徴量情報を取得し、異常検知に使用している。

3.3 実験環境

計測で考える 3 つの連携シナリオについて、図 5 に示す。

エッジ連携シナリオでは、サービス組織がサーバを所有し、自身のデータを管理することを想定している。実験は、2 台のエッジサーバを 1 GbE で接続し計測を行った。送信する情報は Res Net の

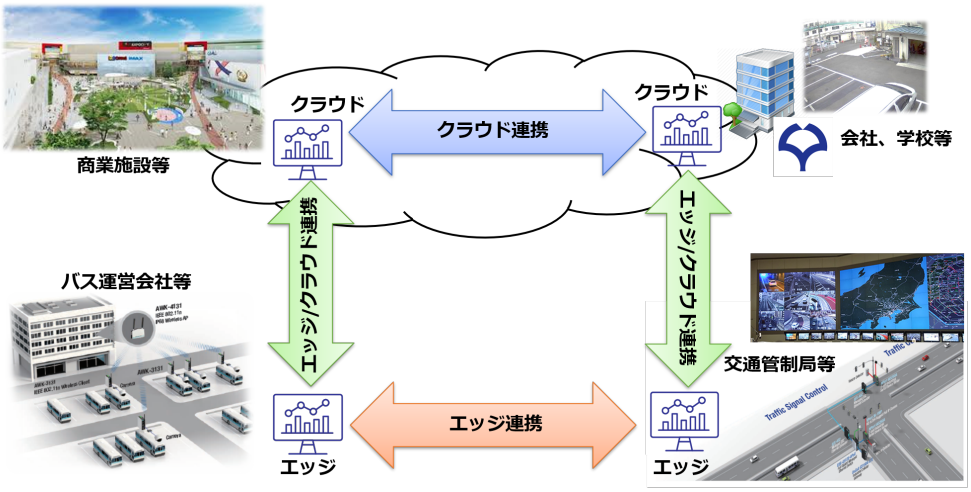

図 5: 3 つの情報連携シナリオ

表 1: 送信側エッジサーバの諸元

使用 OS	Ubuntu 20.10
CPU	AMD Ryzen Threadripper 3970X 32-Core Processor 2.20 GHz (32 コア)

表 2: 受信側エッジサーバの諸元

使用 OS	Windows 10 Pro for Workstations
CPU	Intel(R) Xeon(R) Silver 4216 CPU @ 2.10 GHz 2.10 GHz (16 コア)

中間層の出力値である。Sender、Receiver は、docker 上で動作している。エッジサーバの諸元について表 1 、表 2 に示す。

クラウド連携シナリオでは、リソース制限をしたサーバを送信側受信側ともにクラウドに配置することを想定している。実験は、AWS (Amazon Web Service) のクラウドサーバを 2 台利用し、計測を行った。また、画像そのものをクラウドに上げて共有するのではなく、Res Net の中間層の出力値を送信することで共有している。

エッジ／クラウド連携シナリオでは、送信側はリソース制限をしたサーバをクラウド上に配置し、受信側はサービス管理組織がサーバを所有することを想定している。実験は、送信側は AWS のクラウドサーバ、受信側はエッジサーバを利用し、計測を行った。送信する情報は Res Net の中間層の出力値である。クラウドサーバの諸元を表 3 に示す。

表 3: クラウドサーバの諸元

クラウドサービス	Amazon AWS
使用 OS	Ubuntu Server 20.04 LTS
仮想 CPU	1 vCPU (2.40 GHz 1 コア)
配置地域	東京

3.4 評価方法

3.2 節で記述したシステムを用いて、プラットフォーム連携における連携遅延を計測する。エッジ連携シナリオ、エッジ／クラウド連携シナリオ、クラウド連携シナリオでの実験を行い、各連携シナリオにおける連携遅延の計測を行った。ここで、送受信側で連携し異常検知の処理を行う流れを図 6 に示す。

それぞれの処理の内容を以下に示す。

1. 送信サーバが画像から画像情報取得
2. 送信サーバが接続先を指定
3. 送信側の Context Broker がサブスクリプションを作成し接続確立を連絡
4. 送信サーバが画像情報をエンティティに付与
5. 送信側の Context Broker がエンティティを作成し送信
6. 受信したエンティティに付与されている画像情報を取得
7. 受信サーバが異常の有無を判断

それぞれの連携環境において、エンティティを作成し、送信を行ってから、受信側に新たなエンティティが追加されるまでの時間 (エンティティ作成・通信時間) と、受信側に新たなエンティティが追加されてから画像処理を行う時間 (画像処理時間) と、画像処理の結果から異常の有無を判断し判断結果を出力するまでの時間 (判断時間) の計測を行う。これにより、NW 遅延の観点と処理遅延の観点の両方からプラットフォーム連携の遅延を計測する。

また処理遅延は、処理量によって変化することが考えられる。よって、データ収集間隔、読み込む画像の枚数を変化させ計測を行うことで、プラットフォーム連携の性能評価を行う。

3.5 評価結果

それぞれのプラットフォーム連携における連携遅延計測を行った結果を記述する。また、そこから考えられる課題についても記述する。

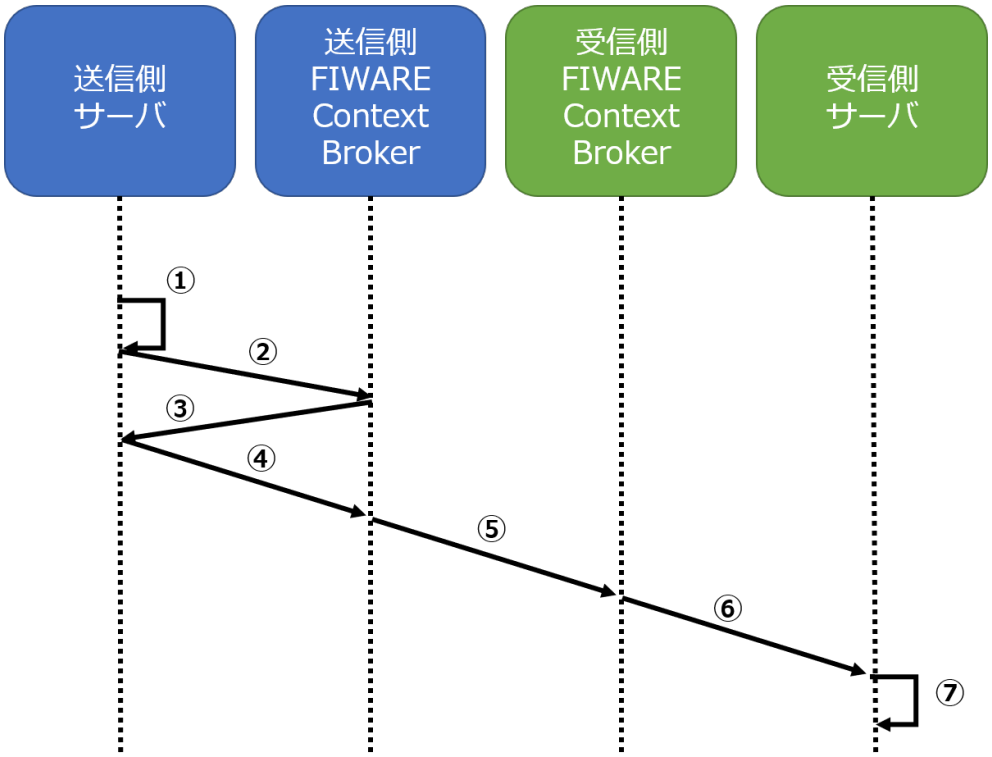

図 6: プラットフォーム連携による異常検知の流れ

3.5.1 エッジ連携シナリオ

まず、送信側受信側両方のエンティティリストが空の状態で、時間の計測を行った。

計測の結果、連携遅延は約 470 ms となった。内訳はエンティティ作成・送信時間が約 105 ms、画像処理時間が約 345 ms、異常判定時間が約 20 ms である。

次に、データ収集間隔を変化させるために、エンティティが 1 秒に 1 個エンティティリストに格納されるとして、エンティティリストの中に 0 個、1 分分 (60 個)、1 時間分 (3600 個)、 6 時間分 (21600 個)、24 時間分 (86400 個) のエンティティが入っている状態で、時間計測を行った。時間計測の結果を表 4 に示す。

計測結果より、エンティティリスト内の個数が増加すると、新たなエンティティ作成・送信までにかかる時間が増加していくことがわかる。また、エンティティ作成・送信にかかる時間が増加したのみの変化で、その他の時間の変化はなかった。このことより、数分や数時間のデータ収集間隔でエンティティを扱う場合は、エンティティ作成・送信の時間に大きな遅延が発生しないと考えられる。しかし、エンティティリスト内の個数を増加させるとさらに作成に時間がかかってしまうことが考えられるため、数日のデータ収集間隔でエンティティを扱う場合は、発生する遅延をより考慮する必要があると考えられる。

表 4: データ収集間隔の変化に対する連携遅延の変化: エッジ連携シナリオ

リスト内の個数 [個]	エンティティ 作成・送信時間 [ms]	画像処理時間 [ms]	異常判定時間 [ms]
0	104	345	20
60	105	345	20
3600	109	345	20
21600	130	345	20
86400	224	345	20

表 5: 画像データ量変化に対する連携遅延の変化: エッジ連携シナリオ

画像 [枚]	エンティティ 作成・送信時間 [ms]	画像処理時間 [ms]	異常判定時間 [ms]
1	104	328	20
5	104	592	100
10	104	965	184
50	105	3255	455
100	109	6427	910

次に、読み込む画像の枚数を増加させていったときの時間計測を行った。なお、エンティティ作成前のエンティティリストの中身は空にしている。計測結果を表 5 に示す。

計測結果より、読み込む画像の枚数を増加させると、画像処理時間が増加していくことがわかる。また、読み込む枚数を増加させていくと、エンティティ作成・送信時間も増加していくことがわかる。このことより、読み込む画像の枚数を増加させると、連携遅延がデータ収集間隔を変化させたときの増加よりも大きく増加すると考えられる。また読み込む画像の枚数が増え、エンティティに付与する情報量が増加すると、エンティティ作成・送信にかかる時間が増加することが考えられる。

3.5.2 エッジ／クラウド連携シナリオ

送信側のプラットフォームをクラウドに配置し、受信側のプラットフォームをエッジに配置した状態で時間計測を行った。なお、送信側受信側両方のエンティティリストが空の状態で、時間の計測を行った。

計測の結果、連携遅延は約 475 ms となった。内訳はエンティティ作成・送信時間が約 110 ms、画像処理時間が約 345 ms、異常判定時間が約 20 ms である。

表 6: データ収集間隔の変化に対する連携遅延の変化: クラウド連携シナリオ

リスト内の個数 [個]	エンティティ 作成・送信時間 [ms]	画像処理時間 [ms]	異常判定時間 [ms]
0	110	880	20
60	111	880	20
3600	118	880	20

計測結果より、送信側をクラウドに配置することにより、エンティティ作成・送信時間が変化したことがわかる。この理由として、エッジから送信を行った場合とクラウドから送信を行った場合の通信遅延が考えられる。またエンティティ作成をクラウドで行っているために、リソース制限による処理能力の差によって、エンティティ作成時間が変化したことも考えられる。

3.5.3 クラウド連携シナリオ

まず、送信側受信側両方のプラットフォームをクラウドに配置し、送信側受信側両方のエンティティリストが空の状態で、時間の計測を行った。

計測の結果、連携遅延は約 1010 ms となった。内訳はエンティティ作成・送信時間が約 110 ms、画像処理時間が約 880 ms、異常判定時間が約 20 ms である。

計測結果より、画像処理時間が大幅に増加したことがわかる。この理由は、クラウドのリソース制限のためだと考えられる。

次に、データ収集間隔を変化させるために、エンティティが 1 秒に 1 個格納されるとして、エンティティリストの中に 0 個、1 分分 (60 個)、1 時間分 (3600 個) のエンティティが入っている状態で、時間計測を行った。時間計測の結果を表 6 に示す。そして、エンティティリスト内の個数を変化させたときのエンティティ作成・送信時間からエンティティ送受信を行う時間を引いたエンティティ作成時間の変化について、エッジ連携シナリオ、エッジ／クラウド連携シナリオ、クラウド連携シナリオでの比較を図 7 に示す。

図 7 より、エンティティリスト内の個数を増加させていくと、エンティティ作成時間はどの連携シナリオでも増加していくことがわかる。またクラウドを用いた連携シナリオよりも、エッジ連携シナリオのほうがエンティティ作成時間が総じて短いことがわかる。このことより、よりデータ収集間隔の大きい情報を扱うにはエッジ連携シナリオが向いているといえる。

またエッジ／クラウド連携、クラウド連携では、エンティティリスト内の個数が多くなると、遅延にばらつきが生じる。このことより、データ収集間隔の大きい情報を扱う場合、遅延に $\pm$ 2 ms の違いを考慮する必要があると考えられる。

次に、読み込む画像の枚数を増加させていったときの時間計測を行った。なお、エンティティ作成

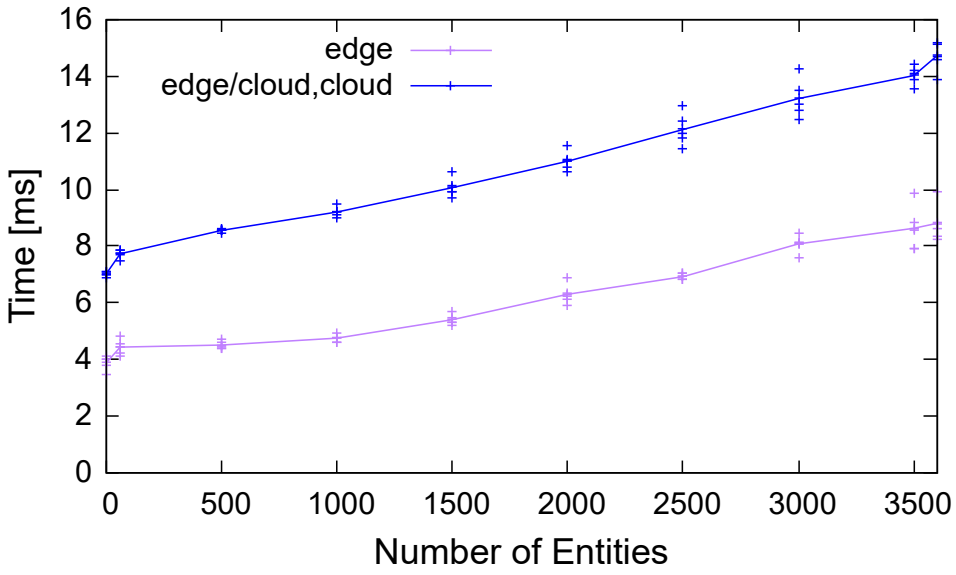

図 7: エッジ連携とクラウド連携の比較 (エンティティ作成時間)

表 7: 画像データ量変化に対する連携遅延の変化: クラウド連携シナリオ

画像 [枚]	エンティティ 作成・送信時間 [ms]	画像処理時間 [ms]	異常判定時間 [ms]
1	110	880	20
5	110	1266	100
10	111	1870	184

前のエンティティリストの中身は空にしている。時間計測の結果を表 7 に示す。そして、読み込む画像の枚数を変化させたときの画像処理時間の変化について、エッジ連携シナリオ、エッジ／クラウド連携シナリオ、クラウド連携シナリオでの比較を図 8 に示す。

図 8 より、読み込む画像の枚数を増加させると、各連携で読み込んだ画像の画像情報を取得する時間が増加していくことがわかる。このことより、読み込む画像の枚数を増加させると、連携遅延がエンティティリスト内の個数を増加させた場合よりも大きく増加すると考えられる。またクラウド連携の場合、画像情報を取得する時間がエッジ連携の場合と比較して長いことがわかる。このことより、クラウド連携で時間粒度の小さい情報を扱うとなると、少ない画像枚数しか扱うことができないと考えられる。またクラウド連携は読み込む画像の枚数が増えるほど画像処理時間の増加量が上昇することより、時間粒度が小さく、読み込む画像の枚数が多い情報を扱うのは厳しいと考えられる。

またクラウド連携では、読み込む画像の枚数が増加していくと、遅延にばらつきが生じる。このことから、データ量の大きい情報を扱う場合、遅延に $\pm$ 100 ms の違いを考慮する必要があると考え

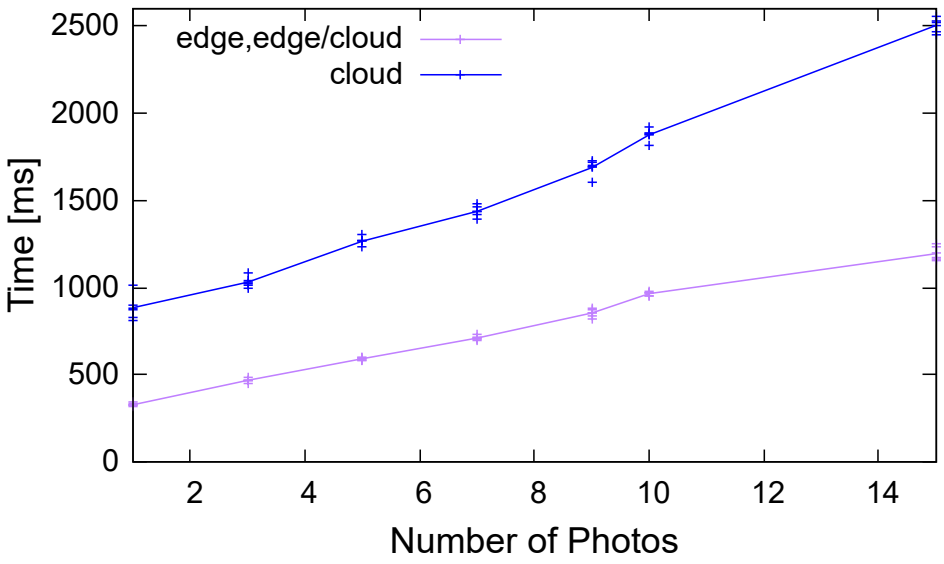

図 8: エッジ連携とクラウド連携の比較 (画像の枚数)

られる。

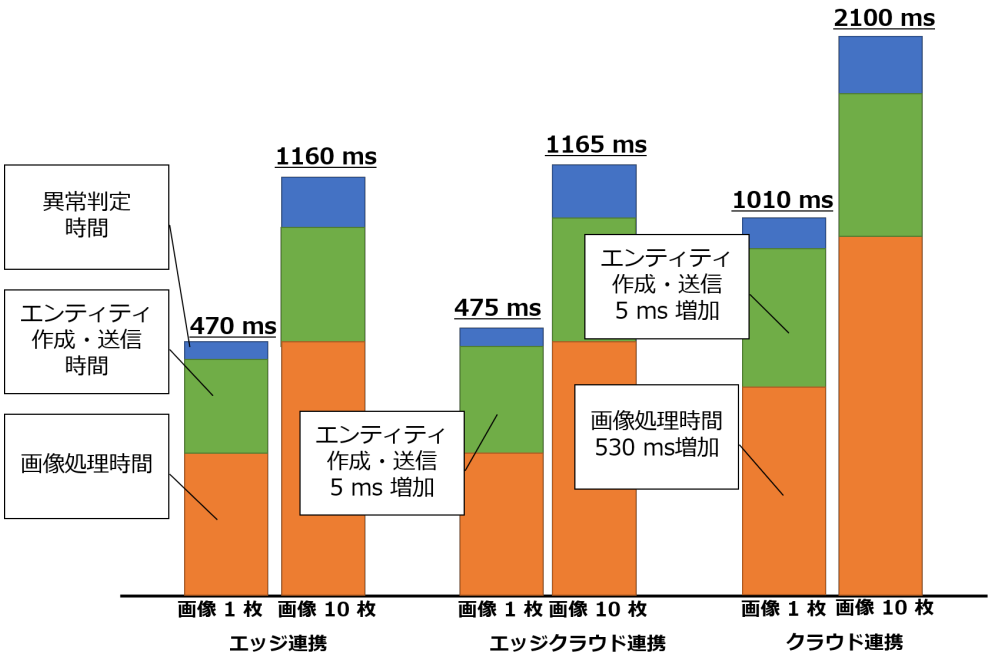

図 9: 各情報連携シナリオにおける連携遅延の比較

4　性能評価結果に基づく IoT プラットフォーム連携の適用領域

前章の性能評価結果により得られる処理データ量やクラウド／エッジ環境に対する遅延特性にもとづいて、IoT プラットフォーム連携の適用領域を整理する。

4.1　センシングデータの時空間特性

ユースケースには、車両の動きや不審者の動きなどの数秒単位で状況が変化する情報を扱うユースケースから、建造物の情報など数日・数年経過しても状況が変化しない情報を扱うユースケースまで、様々な時間粒度の情報を扱うユースケースが存在する。またユースケースには、建物全体の情報や 1 つの地区すべての道路情報など複数のカメラを用いて広範囲からセンシングデータを収集し、その情報を扱うユースケースから、建物や道路の 1 部分などの狭範囲からセンシングデータを収集し、その情報を扱うユースケースまで、様々な空間粒度の情報を扱うユースケースが存在する。

4.2　各連携シナリオの適用領域の整理

まず、読み込む画像枚数を 1 枚、10 枚としたときにエンティティを 1 個作成するときの、各情報連携シナリオにおける連携遅延の比較を図 9 に示す。

図 9 より、クラウド連携シナリオでの連携遅延はエッジ連携シナリオ、エッジ／クラウド連携シ

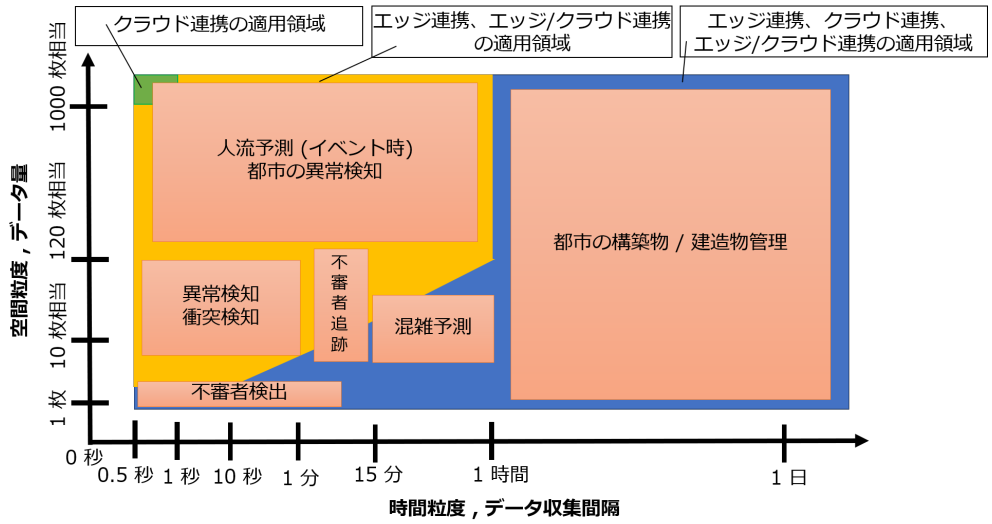

図 10: 各情報連携シナリオにおける適用領域

ナリオでの連携遅延の約 2 倍になっていることがわかる。また、画像枚数を 1 枚から 10 枚に増加させたときの連携遅延の増加量がエッジ連携シナリオでは約 700 ms であるが、クラウド連携シナリオでは約 1000 ms であることがわかる。このことより、クラウド連携シナリオはデータ量が増加するほど連携遅延が大幅に増加していくと考えられる。

前章の評価結果に基づき整理を行った各連携シナリオのサービス適用領域を図 10 に示す。

図 10 では、連携を図るユースケースの例として、今回の道路上の異常検知に加え、不審者検出、衝突検知、不審者追跡、混雑予測、イベント時の人流予測、都市の異常検知、都市の構築物／建造物管理を考えた。1 分以内の時間粒度の情報を 1 枚の画像で扱うユースケースの例として不審者の検出、1 分以内の時間粒度の情報を 1 枚から 120 枚相当の画像で扱うユースケースの例として衝突検知、1 分から 15 分の時間粒度の情報を 10 枚から 120 枚相当の画像で扱うユースケースの例として不審者追跡、15 分から 1 時間の時間粒度の情報を 1 枚から 10 枚相当の画像で扱うユースケースの例として混雑予測、1 時間以内の時間粒度の情報を 120 枚から 1000 枚以上相当の画像で扱うユースケースの例としてイベント時の人流予測や都市の異常検知、1 時間以上の時間粒度の情報を扱うユースケースの例として都市の構造物、建造物管理を想定している。

図 10 より、クラウド連携ではデータ量の増加による連携遅延の増加の影響により、時間粒度が 10 秒未満の情報は画像 1 枚まで、10 秒以上 15 分未満 の情報は 画像 10 枚相当まで、15 分以上 1 時間未満の情報は画像 120 枚相当までしか扱うことができないと考えられる。そのため、イベント時の人流予測や都市の異常検知、衝突検知、扱う画像枚数が多い時の不審者追跡、扱う画像枚数が多い時の混雑予測のユースケースには適用不可であると考えられる。

対してエッジ連携、エッジ／クラウド連携は、1000 枚未満の画像ならば、どの時間粒度の情報で

も扱うことが可能であると考えられる。そのため、クラウド連携で適用可能な不審者検出、扱う画像枚数が少ない不審者追跡、扱う画像の枚数が少ない混雑予測、都市の構築物／建造物管理のユースケースに加え、クラウド連携で適用不可であったユースケースも適用可能になると考えられる。しかし、画像の枚数が 1000 枚以上となると、時間粒度が 1 秒以下の情報は扱うことができないと考えられる。そうなると、時間粒度が 1 秒以下で 1000 枚以上の画像を扱うイベント時の人流予測や都市の異常検知のユースケースには適用不可であると考えられる。しかし、クラウド連携のリソースを追加すると 1000 枚以上の画像を時間粒度 1 秒以下で利用することができると考えられる。だが、クラウドは従量課金制のため、ユースケースのために組織が費用を払う場合、節約のためにリソースを制限することが考えられる。リソースが制限されると 1000 枚以上の画像を時間粒度 1 秒以下で扱うイベント時の人流予測や都市の異常検知のユースケースは適用不可となり、図 10 の青色部分のユースケースのみ適用可能となる。

5　おわりに

本報告では、IoT プラットフォーム間の情報連携の評価し、IoT プラットフォーム連携のユースケース適用領域の整理に取り組んだ。プラットフォーム連携のユースケースとして道路の異常検知システムのプロトタイプを実装し、そのユースケース上でエッジ連携シナリオ、エッジ／クラウド連携シナリオ、クラウド連携シナリオで連携遅延の比較を行い、各連携シナリオについて、適用領域の整理を行った。その結果、エッジ連携シナリオでの連携遅延とクラウド連携シナリオでの連携遅延の間に約 530 ms の違いが生じ、さらに、空間粒度が大きい情報を扱う場合、扱うことのできる情報の時間粒度が異なることがわかった。この扱うことのできる情報の時間粒度の違いから、エッジ連携シナリオ、エッジ／クラウド連携シナリオではリアルタイムデータを扱うユースケースまで適用可能であるが、クラウド連携シナリオではリアルタイムデータを扱うユースケースは適用不可であることが明らかとなった。

本報告では 2 台のプラットフォーム連携について情報連携の評価を行ったが、シナリオによっては複数の IoT プラットフォームで連携を行うことが考えられる。また、複数の IoT プラットフォームが連携する場合、本報告の実装に用いた Context Broker フェデレーション以外にも、RTPS プロトコルを利用した連携が考えられる。様々な情報連携のプロトコルで生じる連携遅延を明らかにしつつ、連携台数のスケーラビリティを評価することが今後の課題として挙げられる。

謝辞

本報告を終えるにあたり、大阪大学大学院情報科学研究科村田正幸教授には、ご多忙の中多大かつ貴重なご指導を賜りましたこと深く感謝いたします。そして、大阪大学大学院情報科学研究科荒川伸一准教授には、研究の進捗・方針確認や論文執筆指導など、手厚くご指導していただきました。心より厚く感謝申し上げます。また、平素よりご指導いただきました先導的学際研究機構大下裕一准教授並びに大阪大学大学院情報科学研究科小南大智助教授に心より感謝申し上げます。最後に、日々の学生生活を支えてくださった研究室の皆様に感謝の意を表し、謝辞とさせていただきます。

参考文献

[1] 島田　啓一郎, “人工知能に入力する最強の実世界情報を作る～人の眼を超えた先端イメージセンシングの IoT 応用～,” **情報通信審議会　情報通信技術分科会　技術戦略委員会　次世代人工知能社会実装 WG（第 5 回）**, pp. 1–23, Apr. 2017.

[2] A. Benayache, A. Bilami, S. Barkat, P. Lorenz, and H. Tale, “MsM: A microservice middleware for smart WSN-based IoT application,” *Journal of Network and Computer Applications*, vol. 144, pp. 138–154, Jun. 2019.

[3] I. Fatima, A. Anjum, S. U. R. Malik, and N. Ahmad, “Cyber physical systems and IoT: Architectural practices, interoperability, and transformation,” *IT Professional*, vol. 8, pp. 46–54, May 2020.

[4] G. Kousiouris, S. Tsarsitalidis, E. Psomakelis, S. Koloniaris, C. Bardaki, K. Tserpes, M. Nikolaidou, and D. Anagnostopoulos, “A microservice-based framework for integrating IoT management platforms, semantic and AI services for supply chain management,” *ICT Express*, vol. 5, pp. 141–145, Jun. 2019.

[5] I. Fajjari, F. Tobagi, and Y. Takahashi, “Cloud edge computing in the IoT,” *Annals of Telecommunications*, vol. 73, pp. 413–414, Aug. 2018.

[6] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, “Keeping the smart home private with smart(er) IoT traffic shaping,” *Privacy Enhancing Technologies*, pp. 128–148, Mar. 2019.

[7] N. Sivakumar, P. Thiyagarajan, and R. Sandhiya, “IoT based on smart agriculture,” *Scientific and Engineering Research*, vol. 9, pp. 151–153, Apr. 2018.

[8] A. González-Vidal, J. Cuenca-Jara, and A. F. Skarmeta, “IoT for water management: Towards intelligent anomaly detection,” in *Proceedings of IEEE 5th World Forum on Internet of Things (WF-IoT)*, Apr. 2019, pp. 867–872.

[9] “IoT platforms: Vertically versus horizontally layered architecture,” https://www.simfony.com/iot-platforms-vertically-versus-horizontally-layered-architecture/, (Accessed on 01/22/2021).

[10] 安永 尚稔, 中村 文昭, 山岸 祥子, “IoT 時代における垂直統合型データウェアハウス基盤,” **雑誌** FUJITSU, vol. 4, pp. 42–48, Jul. 2015.

[11] “IDM - Introduction - Tour Guide,” https://fiwaretourguide.readthedocs.io/en/latest/security/introduction/, (Accessed on 12/24/2020).

[12] "PEP Proxy - Wilma - Fiware-Pep-Proxy," https://fiware-pep-proxy.readthedocs.io/en/latest/, (Accessed on 12/24/2020).

[13] "Context Broker Federation - FIWARE-Orion," https://fiware-orion.readthedocs.io/en/master/user/federation/index.html, (Accessed on 11/20/2020).

[14] "IDM - Introduction - Tour Guide," https://fiware-tutorials.readthedocs.io/en/latest/fast-rtps-micro-rtps/index.html, (Accessed on 11/16/2020).

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Dec. 2016, pp. 770–778.