「マルチメディア，分散，協調とモバイル
(DICOMO2019) シンポジウム」 令和元年7月

分散MQTT Brokerを活用した Dataflowコンポーネント間通信手法の提案

石原 真太郎[1] 安田 和磨[1] 秋山 豊和[1] 安倍 広多[2] 寺西 裕一[3]

概要：Internet of Things (IoT) デバイスから生成されるストリームデータの処理手順を複数のソフトウェアコンポーネントから成る Dataflow application として定義し，それを展開する Dataflow platform の研究開発が進められている．これまで，P2P ネットワークを用いて，コンポーネントに付与した値を用いて負荷分散する手法が提案されているが，単純なラウンドロビンであり，コンポーネント配備先リソースの状況は考慮できていなかった．本研究では，オーバレイネットワークに参加するノードから収集した集約値を用いた条件付き Multicast を拡張し，ノードの負荷などの情報を考慮して，適切なノードを選択して配送する Anycast 機構を導入し，ノード状況が変動する環境での配送先選択に用いた．また，エンドノードが P2P 機能をもたない単純なデバイスでも利用可能とするため，MQTT プロトコルに対応した分散型の Topic Based Pub/Sub Broker PIQT を拡張する形で条件付き Anycast 機構を実現した．さらに，拡張したコンポーネント間通信機能を活用したアプリケーションの開発を容易にするための Wrapper 機能について検討した．集約値の情報は，収集してからオーバレイ上のルーティングに反映するまでに時間が必要なため，情報の鮮度は更新頻度に依存する．そこで，Anycast メッセージの受信によりリソースの状態が更新される環境で，Anycast 機構の更新頻度とオーバレイ上での無駄なメッセージホップ数の関係をエミュレーションにより明らかにした．結果として，本研究で想定しているバスセンシング環境におけるストリーム処理を対象とした場合には，リソース探索メッセージ数に対して無駄なメッセージ数を十分に低減できることが確認できた．

A Proposal of an Inter Dataflow Component Communication Method using Distributed MQTT Broker

SHINTARO ISHIHARA[1] KAZUMA YASUDA[1] TOYOKAZU AKIYAMA[1] KOTA ABE[2]
YUUICHI TERANISHI[3]

1. はじめに

近年，Internet of Things (IoT) デバイスが普及し，それらから生成されるデータは都市のスマート化へと活用されている．これまで IoT デバイスでセンシングされたデータをクラウドで収集，分析する IoT アプリケーション等が多種多様に構築されている．しかし，文献 [1] では，全世界のモバイルデータトラフィック量は 2017 年から 2022 年の間に約 7 倍増加すると予測しており，クラウドへの処理負荷や物理的な距離による通信遅延などが課題になると述べている．このような課題を解決するため，IoT デバイスの近傍にソフトウェアコンポーネントを設置するエッジコンピューティングの開発が進められている [2][3]．エッジコンピューティングでは，遠隔地に展開されるサーバ群をクラウド，基地局や収容局に設置される小規模のサーバ群をエッジとして，ネットワークを階層的に分割するアーキテクチャとなっている．エッジでデータのフィルタリングや集約といったアプリケーションの一部の処理を行うことによるクラウドへの通信量の削減や，近傍に配置したエッジからデバイスへ直接処理結果を返すことによる通信遅延の低減が期待されている．エッジコンピューティングのネットワークアーキテクチャとして，Multi-access Edge Computing(MEC) [4]，Fog Computing [5]，Cloudlet

1 京都産業大学 先端情報学研究科
2 大阪市立大学大学院 工学研究科
3 情報通信研究機構

[6] などが提案されており，どのアーキテクチャにおいても，ネットワーク階層は 2 から 3 階層で定義されている．本研究では同様な階層化ネットワークを想定し，IoT デバイスが接続されるネットワークを “Device Network”, 基地局や局舎などに設置される小規模なネットワークを “Edge Network”, クラウドに展開される大規模なネットワークを “Cloud Network” と称した 3 階層とする．また，Azure IoT [7] や Google Cloud IoT Edge [8] など，IoT デバイスから生成されるストリームデータの処理手順を複数のソフトウェアコンポーネントを繋ぎ合わせることで定義し，それを展開して実行する Dataflow platform の研究開発が進められている. 文献 [9] と同様に本研究ではこの処理手順を Dataflow application と呼ぶ.

Dataflow application を構成するソフトウェアコンポーネントは，図 1 に示すように，適切なネットワーク階層に配置されることが求められる．通常各ネットワーク階層は，異なる事業者により運用されていることが多く，異なるネットワークにまたがるコンポーネント配置ならびにコンポーネント間接続には，同じ事業者による複数ネットワークの統合管理が必要になる．文献 [9] では，特定のクラウドサービスに依存せず，Pub/Sub 基盤を用いたエッジコンピューティングを考慮した Dataflow platform を提案している．主に，application コンポーネント間通信に着目しており，publish/subscribe 時にインデックス値を付与させることでコンポーネント間の相互接続と配送先を負荷分散させる機能を実現している．しかし，提案されている方式では publish 側がインデックス値を選択する必要があり，負荷状況等に応じて配送先を切り替えるためには，別途 subscribe 側から情報を収集する必要がある．また，コンポーネントの管理方法については言及していない．

これまでに，我々の研究グループでは，オープンソースソフトウェアである OpenStack [11] を用いることで，クラウド事業者やネットワーク機器によるベンダーロックインを回避しながら，複数事業者にまたがるコンポーネント管理方式を提案した [10]．本研究では，分散型の Topic-based Pub/Sub 基盤である MQTT Broker PIQT[12] に実装されている集約値に基づいた Multicast[14] を拡張することで Anycast 機構を実現し，コンポーネント管理手法を考慮しながら，コンポーネントのリソース状況に応じた負荷分散等が可能なメッセージの配送方式の実現を目指す．

以下，2 章では関連研究である文献 [9] について述べる．3 章では，提案するコンポーネント間通信に関して，想定環境および集約値を用いた Anycast 機構の実現方法について説明する．また，コンポーネントへの通信機能追加のための Wrapper の設計について説明する．4 章では，集約計算遅延が及ぼす影響の評価とバスのユースケースにおける評価について述べ，最後に 5 章でまとめと今後の課題を述べる．

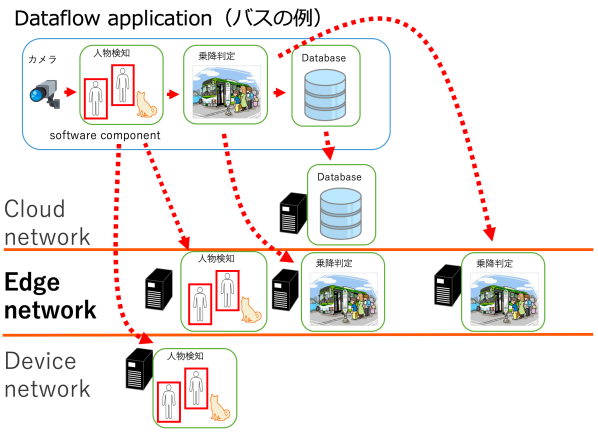

図 1　Dataflow の流れ

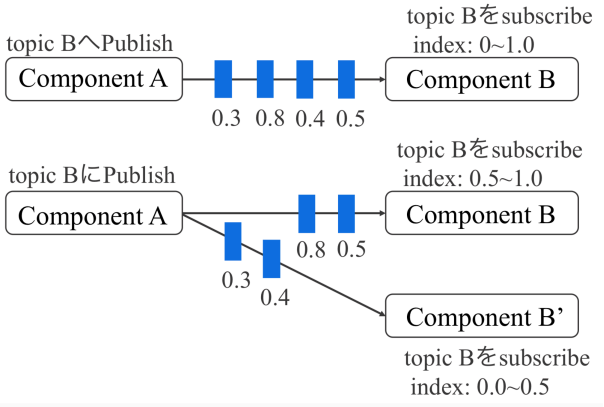

図 2　インデックス値を用いた配送

2. 関連研究

文献 [9] ではエッジコンピューティング環境において，分散型 Pub/Sub 基盤および P2P 構造化オーバレイネットワークである Chord# [15] を活用したコンポーネント間通信手法が提案されている．topic と階層情報の両方を用いてオーバレイネットワークを構築することで，階層化ネットワークを考慮したデータ配送と負荷分散を可能とする動的な Dataflow platform を実現している．図 2 で示すように Pub/Sub 通信方式によりコンポーネント間を繋げており，subscribe 時に topic，階層情報，インデックス値を指定する．インデックス値は 1 つの階層で 0.0〜1.0 の範囲を持っており，同じ階層に所属する同じ topic を subscribe しているコンポーネントではインデックス値の範囲が分割される．publish 時に適切に階層情報とインデックス値を指定することにより，配送先の指定や，負荷分散が可能となっている．下位層のリソース状況を考慮してインデックス値を付与できれば，CPU 負荷などを考慮した配送が実現可能ではあるが，下位層の情報とインデックス値を適切に付与させ管理する必要があり，platform 利用者の負担となる．本研究では，Anycast を用いて，これらの研究課題を解決する．

3. 提案するコンポーネント間通信方式

本章では，提案するコンポーネント間通信方式について説明する．3.1 節では，ターゲットとする Dataflow application に基づいて，Dataflow platform の想定環境を述べ，3.2 節で提案方式の概要について述べる．次に，3.3 節で集約値に基づいた Multicast について述べ，3.4 節で提案する Anycast 機構について述べる．3.5 節では Anycast 機構に伴う PIQT と MQTT v5 を前提としたコンポーネント側とのインタラクションについて述べ，3.6 節ではコンポーネントの実装を容易にするため，提案方式に必要なリソース情報のモニタリング機構などを備えた Wrapper の設計について述べる．

3.1 想定環境

本研究では Dataflow application の例として，バス車内に設置した IoT デバイスからバスの速度や旋回時に発生する乗客の求心加速度を用いた危険運転検知，車両情報を用いた運転手の勤務状態推定，ドライブレコーダの映像を用いた乗降車人数カウントといった Dataflow application を想定している．Dataflow platform では，例えば，映像を用い た対象の動作検出などリアルタイムに前後のデータを比較 する処理を持ち，同一のコンポーネントで解析することが望ましい場合 と，各データごとに独立に処理できる場合がある．同一のコンポーネントで処理する場合，データ配送前に別途リクエストメッセージを送信することで，必要なコンポーネントの確保，データ配送経路を生成し，それに従ってデータを配送する．例えば，ドライブレコーダを用いる Dataflow application の場合は，電源の ON-OFF を実行条件とすることで，電源 ON 時にコンポーネントを確保，配送経路を生成しデータを配送する．電源 OFF 時には確保していたコンポーネントを解放する．一方で，依存関係がなくデータを独立して処理できる場合は経路を生成せずに，データ配送時に活用するコンポーネントを決定することで負荷分散の柔軟性を高める．ここで，対象とするデータの判別，コンポーネントの再利用性を高めるために使用済みコンポーネントの解放，Back pressure 機能によるコンポーネントの故障検知の解消などの様々な機能がコンポーネントには必要となる．各コンポーネントにそれらの通信機能を追加するとサービス開発者への負担やコンポーネントの汎用性の低下を招くため，それらの機能を抽出し，Dataflow コンポーネント用の Wrapper の導入を目指す．

3.2 提案方式の概要

図 3 に提案するコンポーネント間通信方式の概要を示す．提案方式では PIQT を用い，Anycast によって負荷状況に応じて柔軟に配送先を選出する．PIQT は publish/subscribe

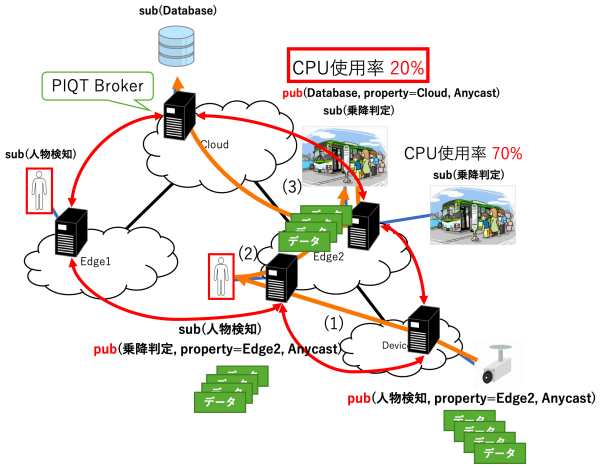

図 3 集約値を用いた配送

通信プロトコルである MQTT [16] に対応した組み込み型 MQTT Broker である Moquette [17] を活用しており，Broker 間通信に P2P 構造化オーバレイネットワークである Suzaku を提供する PIAX [18] を用いている．さらに，Broker には階層情報を保持させることができ，Broker 単位でのネットワーク階層を考慮したデータ配送を可能とする．PIAX では，集約値に基づいた Multicast に対応した Suzaku を提供しており，本研究ではその機構を拡張することで，CPU 使用率などの下位層のリソース状況に応じた Anycast によるコンポーネント間のデータ配送を実現する．図 3 に示すように，MQTT の次期 version である MQTT v5 [19] の property のうち自由に定義することができる User Property を活用して，publish 時に階層情報や配送方法を指定することで，特定の階層への配送を可能とする．本研究で提案する通信方式を利用可能とするために，IoT 分野で標準的に利用されているプロトコルとの互換性を考慮して提案する通信方式を導入するための検討事項を整理した．本稿では特に，MQTT v5 を対象に，提案方式の実装方法を検討する．

3.3 Suzaku と集約値

この節では，PIAX が提供する Suzaku の集約値について述べる．Suzaku は Chord$^{\#}$ を拡張したアルゴリズムである．以下では，はじめに 3.3.1 節で Chord$^{\#}$ について述べ，3.3.2 節で Chord$^{\#}$ を拡張した Suzaku，3.3.3 節で集約値について述べる．

3.3.1 Chord$^{\#}$

Chord$^{\#}$ は key 順序型構造化オーバレイの代表例である．各ノードは key を持っており，リング状に key を辞書順で整列した状態で維持される．key の検索を効率化するために各ノードは他のノードへのポインタを保持し，自ノードの key の次に大きな key を持つノードを Successor，次に小さい key を持つノードを Predecessor と呼び，その他に複数のノードへのポインタを持つ FingerTable(FT) を保持

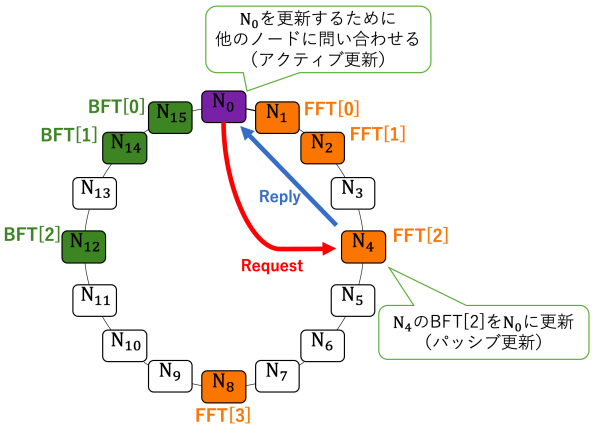

図 4　Suzaku の概要と更新

する．FT に該当するノードは下記の式で求められ，$FT[i]$ は 2^i 個先のノードの情報とそのノードがもつ key 値からなるポインタをもつ．

$$FT[i] = \begin{cases} Successor & (i = 0) \\ FT[i-1] & \\ \quad \rightarrow getFinger(i-1) & (i \neq 0) \end{cases} \quad (1)$$

式 1 の $FT[i]$ は FT に登録されているノードを指し，$FT[0]$ には Successor が登録される．あるノードを p, ノード p の $FT[i-1]$ が指すノードを q とすると，$i > 0$ の場合，ノード p の $FT[i]$ には，ノード q の $FT[i-1]$ に登録されているノード対して $getFinger(i-1)$ を実行して，取得したノードが登録される．あるノードの $FT[i]$ $(i > 0)$ の更新は定期的に行われ，$FT[1]$ から順番に定期的な時間間隔で更新される．検索の時に FT を用いることでノードの総数を n(ただし n は自然数) とすると，FT 収束時の最大ホップ数は $\lceil \log_2 n \rceil$ で収束する．

3.3.2　Suzaku

Suzaku の Chord$^\#$ に対する大きな変更点は FT を双方向に持ち，各ノードが Chord$^\#$ での時計回り方向にノードを指す Forward Finger Table(FFT) の他に，反時計回り方向にノードを指す Backward Finger Table(BFT) が追加されたことである．FFT の段数はノードの総数が n (ただし n は自然数) の場合，$2^{k-1} < n$ を満たす k の最大の整数となり，FT は更新処理により鮮度が保たれる．FT の更新は，ノード挿入時，削除時に行われるが，それ以外にもノードの故障などで不整合が生じた場合，ポインタを修正するために定期的に行われる．ノード挿入時には $FFT[0]$, $BFT[0]$, $FFT[1]$, $BFT[1]$, $FFT[2]$, ..., $FFT[k]$, $BFT[k]$ の順に更新要求のための Request を送信し，更新が行われ，削除時には削除ノードを指し示すノードへと通知し，通知内容に含まれた代用のノードへとポインタを付け替えることで更新する．定期的な更新では，$FFT[1]$, $FFT[2]$, ..., $FFT[k]$ の順に指定された時間間隔で Request を送信し，更新する．次に，図 4 を用いて，N_0 の $FFT[2]$ の更新

表 1　N_0 の FT

FT	集約範囲	集約値
$FFT[-1]$	$[N_0, N_1)$	0
$FFT[0]$	$[N_1, N_2)$	1
$FFT[1]$	$[N_2, N_4)$	3
$FFT[2]$	$[N_4, N_8)$	7
$FFT[3]$	$[N_8, N_0)$	15
$BFT[0]$	$[N_{15}, N_0)$	15
$BFT[1]$	$[N_{14}, N_{15})$	14
$BFT[2]$	$[N_{12}, N_{14})$	13

例を説明する．図 4 のノード数は 16 であり，各ノードは N_i を key として持つとする．更新には，アクティブ更新とパッシブ更新の 2 種類がある．アクティブ更新では，N_0 は $FFT[2]$ に Request を送信し，Reply により N_4 を取得することで更新する．パッシブ更新では，Request を受信した N_4 の $BFT[2]$ に N_0 で更新する．

Suzaku では，範囲検索により，センサデータの効率的な取得や，範囲検索を用いたメッセージの配送を実現する．メッセージの配送例として，N_0 から N_2, N_3, ...,N_8 の全てのノードにメッセージを配送する場合，検索する範囲として $[N_2, N_9)$ を生成し，範囲検索を行う．$[N_2, N_9)$ は $N_2 \leq$ key $< N_9$ を意味する．N_0 は生成した検索範囲を FT が指すノードを元に分割して，各ノードに検索を委譲する．この場合，N_2 が担当する範囲は $[N_2, N_4)$，N_4 が担当する範囲は $[N_4, N_8)$, N_8 が担当する範囲は $[N_8, N_9)$ となる．N_2, N_4, N_8 では担当した範囲で再帰的に範囲検索が行われ，最終的に検索範囲内に該当する全てのノードの結果が N_0 に集約される．

3.3.3　集約値

文献 [14] では，ノードが保持する集約値に基づく条件付き Multicast を提案している．集約値とは各ノードに集約対象の値 (TV) をサービス開発者が定義した集約方法で集めた値である．条件付き Multicast は，グリッドのような分散システムにおいて指定した範囲内で CPU 負荷が最小のノードを選択する場合や，閾値を超えたセンサデータを指定した範囲内から取得する場合に応用できる．PIAX では，文献 [14] に基づいた条件付き Multicast に対応した Suzaku が実装されており，配送先を効率よく決定するために予め一定の範囲のノードの値を集約値として，FT 上に保持する．文献 [14] では，集約値を短時間で収束可能な FT の更新方式 (連続的更新方式) も提案されているが，PIAX では提案された方式は実装されておらず，定期的に更新する方式 (離散的更新方式) しか実装されていない．以降ではすでに実装されている離散的更新方式を想定して議論を進める．

条件付き Multicast に対応した Suzaku での集約範囲の決定方法および集約値の更新方法について述べる．Suzaku には，$FFT[i]$ および $BFT[i]$ に Chord$^\#$ でのポインタに

加えて，複数ノードを含む範囲である集約範囲と，集約範囲内に含まれるノードの値を定義された集約方法で計算した集約値が含まれる．$FFT[i]$ には，$FFT[i]$ のポインタが指すノードから時計回りに 2^i ノードを含む集約範囲と集約値を登録される．$BFT[i]$ にはトポロジがリングの関係上，時計回りに 2^{i-1}(ただし，$i=0$ の場合は 1) ノードを含む集約範囲と集約値を登録される．表 1 は図 4 の N_0 が保持する FT を示し，各 FT が保持する集約範囲と集約値の例である．例では，各ノードが保持する集約値をノードの key に含まれる数字とし，集約方法は数字の最大値とする．表 1 の $FFT[1]$ の場合，$FFT[1]$ の集約範囲に含まれるノードは N_2, N_3 であり，各ノードは key に含まれる数字の 2, 3 を集約値として保持する．$FFT[1]$ にはこれら 2 つの値の最大値となる 3 が集約値として登録される．また，FT には自ノードを指す $FFT[-1]$ が存在し，$FFT[-1]$ には自ノードのみを含む集約範囲と自ノードが保持する値を集約値として登録される．集約値は FT の更新で同時に更新するため，集約値の鮮度は FT 更新の頻度に依存する．例えば，N_0 が $FFT[i]$ のアクティブ更新を行う時，N_0 は送信先のノードのパッシブ更新用に $FFT[i-1](i>0)$ までの範囲とその範囲内の各ノードから集約値を計算する．そして，その範囲と計算した集約地を Request に含めて，$FFT[i]$ に送信する．Request の受信ノードは自ノードから時計回りに 2^i ノードを含む集約範囲と集約した値を N_0 に Reply として送信する．また，受信ノードは Request に含まれたパッシブ更新用の情報を用いて，$BFT[i]$ のパッシブ更新を行う．ここで，$FFT[i]$ の集約値の更新には $FFT[1]$ から $FFT[i-1]$ $(i>0)$ までの集約値を用いるため，正しく $FFT[k]$ を更新するには，1 段ずつ順番に更新する必要がある．FT 更新の周期を T 秒とすると，あるノードが FT の集約値を更新するには 1 段につき最悪 T 秒掛かるため，kT 秒必要となる．さらに，自身の集約値を正しく更新するには，参照する他ノードの集約値が正しく更新されている必要がある．例えば，N_0 の保持する値に何らかの更新があった場合，N_0 の $FFT[2]$ の集約値を更新するには，N_4 の $FFT[1]$ までの集約値が更新されていなければいけない．この時，N_4 の更新がまだ $FFT[0]$ であった場合，N_0 の $FFT[2]$ の集約値を正しく更新するには N_4 での集約値の更新が順次実行され，N_4 の $FFT[1]$ までの集約値の更新が完了する必要がある．従って，kT 秒では全ノードの集約値は 1 段分しか更新されない．よって，FT の全段の集約値は最悪 k^2T 秒掛ければ正しく更新される．これを計算量で表すと，$O(T(\log n)^2)$ となる

3.4 集約値を活用した Anycast 機構の提案

この節では，3.3.2 節で述べた Suzaku を PIQT で活用した場合に形成されるオーバレイネットワークについて 3.4.1 節で述べた後，3.4.2 節で提案する Anycast 機構について述べる．

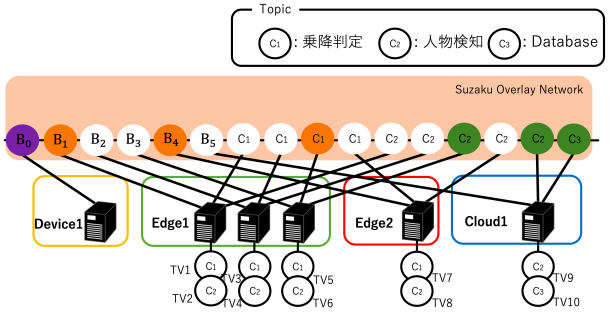

図 5 物理と Overlay の関連性

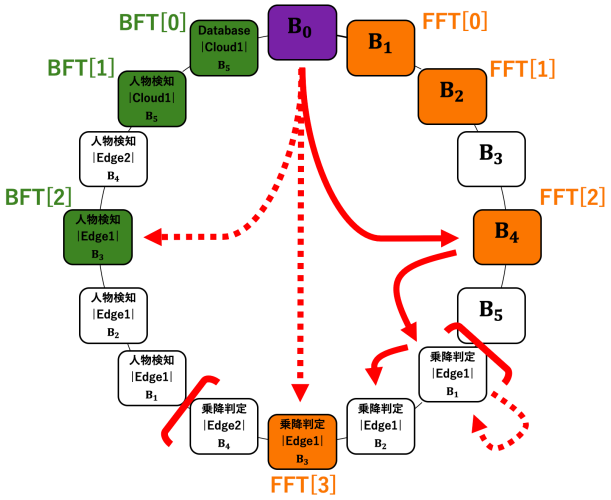

図 6 提案するルーティング

3.4.1 PIQT で用いる Suzaku

PIQT で形成される Suzaku のオーバレイネットワークでは，Broker 接続時と topic の subscribe 時にノードが生成される．Broker 接続時にはノードは Broker の ID を key として保持する．topic の subscribe 時にはノードは topic と階層情報，接続している $BrokerID$ を | で繋げた “topic|階層情報 |$BrokerID$” を key として保持する．生成した key により，ノードはオーバレイ上に Broker の key をリングの始めにまとめ，その後に，| で繋げられた topic, 階層情報, $BrokerID$ の順番に辞書順で整列される．また，Broker には複数の subscriber を接続すること可能であり，同じ topic を持つ複数の subscriber は，1 つのノードとしてまとめられるが，異なる場合，新たにノードを生成する．従って，1 つの Broker は複数の key を保持できる．

例えば，図 5 に示すように Cloud1. Edge1, Edge2, Device1 から構成される階層化ネットワークを想定し，Dataflow application コンポーネントとして乗降判定，人物検知，Database を各ネットワーク階層に配置する．この場合，PIQT では，前段落で述べたことを踏まえて，図 6 に示すようなオーバレイネットワークが構築される．PIQT では topic へのメッセージの配送を Suzaku での範囲検索により実現している．生成された $BrokerID$ の最小を

$BrokerID_{min}$，最大を $BrokerID_{max}$ とすると，B_0 から Edge1 の階層に存在する乗降判定を持つ全てのノードにメッセージを配送する場合，[乗降判定 | $Edge1$ | $BrokerID_{min}$, 乗降判定 | $Edge1$ | $BrokerID_{max}$) が検索範囲として生成される．B_0 は，検索範囲を B_4 が担当する範囲を [乗降判定 |$Edge1$|$BrokerID_{min}$, 乗降判定 |$Edge1$|B_3)，乗降判定 | $Edge1$ | B_3 が担当する範囲を [乗降判定 |$Edge1$|B_3, 乗降判定 |$Edge1$|$BrokerID_{max}$) とし，各ノードにメッセージの配送を委譲する．各ノードは担当した範囲で再帰的に範囲検索が行われ，乗降判定 |$Edge1$| を含むノード全てにメッセージを配送する．

3.4.2 集約値を活用した Anycast 機構

本節では集約値を活用した Anycast 機構について述べる．Anycast 機構の配送手順を以下に示す．

(1) FT から配送候補のノードを選出
検索範囲と FT に登録された各ノードが保持する集約範囲が重なる場合，そのノードを配送候補として選出する．

(2) 配送ノード選出
複数の配送候補から 1 つのノードを選出する．ノードの選出ロジックは，記述可能になっており，最小値，最大値，ランダムなどに変更することができる．

(3) メッセージを配送
(3) は以下の 3 つケースに分岐する．

(a) 選出ノードが他ノードの場合
範囲検索と同様に，選出ノードに担当範囲を割り当てメッセージを送信する．ここで，選出ノードの担当範囲以外の範囲は検索から除外する．

(b) 選出ノードが自ノードの場合
選出ノードにメッセージを配送し，検索を終了する．

(c) 選出ノードが存在しない場合
集約値の誤った情報により，検索範囲に該当するノードが存在しない場合，送信者の FT から検索範囲に一番近いノードにメッセージを配送する．

上記の配送手順において，(3)(a), (3)(c) の場合，(1) に戻り，最終的に (3)(b) が処理されるまで再帰的に繰り返される．

表 2 は図 6 の B_0 が保持する FT を示しており，各 FT の集約する範囲と集約値の一例である．例では，図 5 のように各コンポーネントが保持する TV を処理可能なコンポーネント数とし，集約値では topic | 階層情報 | に処理可能なコンポーネント数を結びつけた形て保持しており，集約する方法は同じ topic | 階層情報 | なら数を足し合わせている．図 6 では B_0 から Edge1 の乗降判定を持つノードのどれか 1 つにメッセージを配送する場合を示しており，B_0 から [乗降判定 |$Edge1$|, 乗降判定 |$Edge1$|) の検索範囲で範囲検索を活用したメッセージの配送を行う．はじめに，B_0 の FT である表 2 から (1) の処理より，B_4, 乗降判定 |$Edge2$|B_3 の 2 つが選出される．その後，(2) の処理で選出ロジックを適用し，B_4, 乗降判定 |$Edge2$|B_3 のノード数が同じであるため，ランダムで B_4 が選出されたとし (3)(a) の処理で検索範囲を [乗降判定 |$Edge1$|, 乗降判定 |$Edge1$|$B3$) に縮小して B_4 に委譲する．この一連の処理を再帰的に繰り返すことで，図 6 では最終的に乗降判定 |$Edge1$|B_2 にメッセージを配送する．一方で，集約値の更新は FT の更新に依存するため，集約値には間違った値が入っている可能性がある．

表 2　B_0 の FT

FT	集約範囲	集約値
$FFT[-1]$	[B_0, B_1)	null
$FFT[0]$	[B_1, B_2)	null
$FFT[1]$	[B_2, B_4)	null
$FFT[2]$	[B_4, 乗降判定 |Edge1|B_3)	乗降判定 |Edge1| : 2
$FFT[3]$	[乗降判定 |Edge1|B_3, B_0)	乗降判定 |Edge1| : 1 乗降判定 |Edge2| : 1 人物検知 |Edge1| : 3 人物検知 |Edge2| : 1 人物検知 |Cloud1| : 1 Database|Cloud1| : 1
$BFT[0]$	[Database|Cloud1|B_5, B_0)	Database|Cloud1| : 1
$BFT[1]$	[人物検知 |Cloud1|B_5, Database|Cloud1|B_5)	人物検知 |Cloud1| : 1
$BFT[2]$	[人物検知 |Edge1|B_3, 人物検知 |Cloud1|B_5)	人物検知 |Edge1| : 1 人物検知 |$Edge2$| : 1

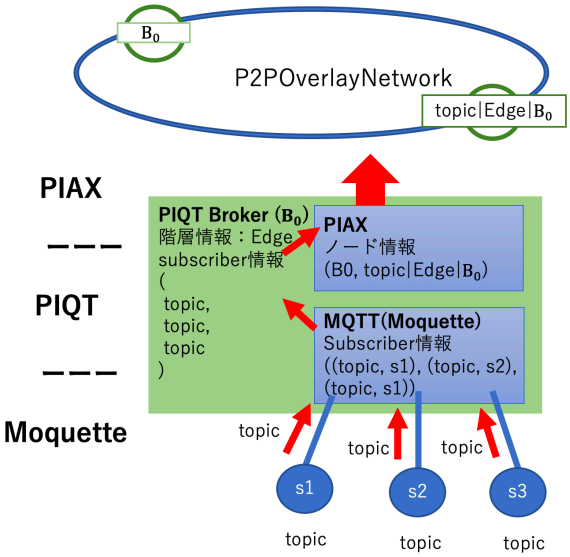

図 7　subscriber 情報の収集

3.5 Anycast に伴う PIQT の機能拡張

3.4 節で提案した配送手法では，集約値に CPU 使用率などの TV を載せることで，CPU 使用率を考慮した配送を実現することが可能になる．本節では，CPU 使用率などのリソース状況に応じた Anycast を実現する上での PIQT の機能拡張について述べる．

図 7 に Moquette, PIQT, PIAX を用いて，MQTT クライアントから topic の情報を収集する流れを示す．MQTT クライアントの状態は Moquette で管理し，subscriber の

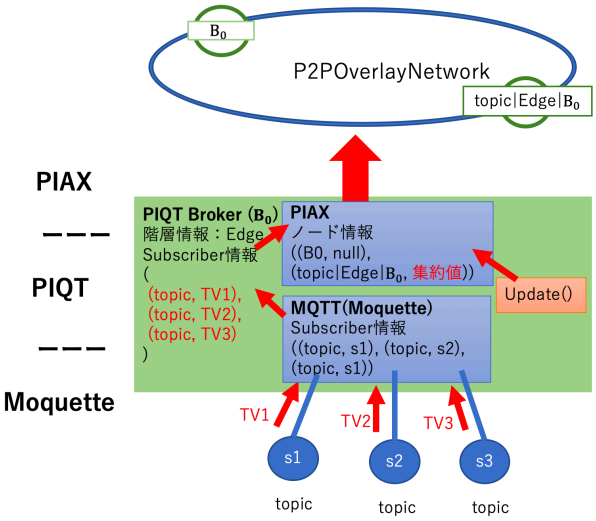

図 8　PIAX への PIQT の拡張

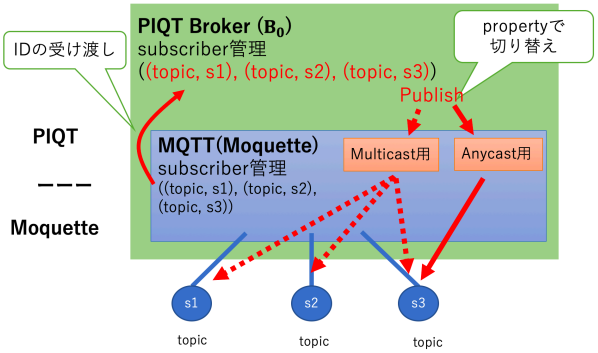

図 9　PIQT への subscriberID の通知

情報は PIQT で集約したあと，その情報を PIAX のオーバレイに登録する．PIAX 上に実装された Suzaku の集約機能を PIQT で活用するためには，MQTT クライアントから *TV* を PIQT で収集し，PIAX 側と連携させる必要がある．図 8 に PIQT の実装と PIAX 側との連携箇所において，*TV* を収集する流れを示す．前述したように，PIQT では subscriber の情報を収集する．subscriber から収集する各 *TV* も同様に，PIQT で収集し，各 topic に結びつけて管理する．収集した *TV* はサービス開発者が指定した集約方法で計算し，集約値として PIAX のオーバレイに登録するように実装した．また，*TV* として，CPU 使用率といったモニタリング情報も対象にしており，それらは動的に変動するため，PIAX 上で登録した値の鮮度を維持するには更新処理が必要となる．そこで，PIQT に update() を実装し，オーバレイ上の集約値の更新処理を実現する．更新処理は FT の更新処理を模倣し，subscriber の参加および，脱退時や定期的な時間間隔で更新させる．

次に PIQT から subscriber へメッセージを配送する方法について説明する．既存の実装では PIQT から topic とデータを Moquette に渡すことで，Moquette により，その topic に該当する各 subscriber へデータが配送される．

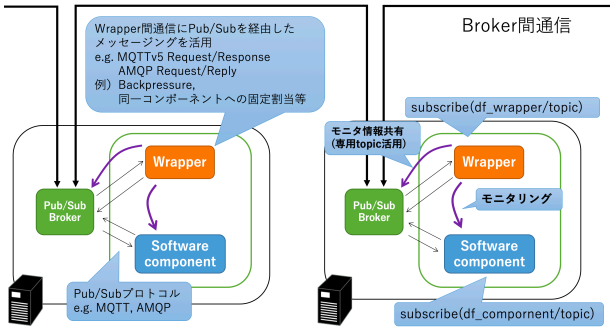

図 10　Wrapper の設計

この場合，topic ベースの Multicast は実現されるが，同一 topic 内で特定の配送先を選択する Anycasat ができない．これを解決するために，図 9 に示すように，PIQT と Moquette を拡張した．Moquette ではコネクション時に生成される subscriber ごとにユニークな ID と topic を結びつけて subcriber の情報が管理されるが，Moquette から PIQT には topic のみが通知される．そのため，topic が同じ subscriber はまとめられ，PIQT では各 subscriber の情報が取得できない．図 9 では，コネクション時に生成する ID を PIQT に渡し，Moquette と同様の方法で保持させる．さらに，Moquette に用意されている配送機構を Multicast 用としてそのまま扱い，PIQT に保持させた ID を考慮して特定の subscriber に配送する機構を Anycast 用として Moquette に追加した．PIQT ではこれら 2 つの配送機構を publish 時の property を指定することで切り替える．

3.6　Wrapper の設計

この節では，想定する Dataflow platform に導入する Wrapper 機構の設計について述べる．

3.5 節で述べたように，拡張した機能を活用するには，subscriber から *TV* を Broker に送信する必要がある．また，3.1 節で述べたように，コンポーネント群を配置するリソースを予め占有することで，一連の Dataflow に同じリソースを割り当てることが可能になる．Dataflow 処理では前後関係を考慮することを必要とする場合があり，コンポーネントの間での調整が発生する．本研究では，Flink [20] , NodeRED [21] , Fluentd [22] といった既存の Dataflow platform のコンポーネントを再利用可能とするため，本研究で提案する platform で要求される機能をコンポーネントと Broker を中継する Wrapper として実装する．Wrapper は図 10 に示すように，コンポーネント間の通信を中継する．ユーザがある topic のコンポーネントを作成したときに，図 10 に示すように，デプロイ時にコンポーネントには df_compornent/topic，Wrapper には df_wrapper/topic のように topic を変換して，PIQT へ subscribe させることで，Broker 経由でのコンポーネントと Wrapper 間の通信を可能にする．また，TV の登録には別途専用の topic

を用意する．上述のように topic を変換することで，一旦 Wrapper 上で TV を収集した上で，PIQT に登録する．ここで，Dataflow application コンポーネントの予約や Back pressure 機能を実現するには Wrapper 間で適切に通信することが求められる．例えば，コンポーネント予約メッセージに対する予約完了メッセージ，Back pressure に関する送信元コンポーネントへのメッセージおよびそれに対する応答や，生存確認と応答など，特定のメッセージに対する応答処理が必要となる．これを MQTT v5 の Request/Response を活用することでデータの送信者の Wrapper に通知することを実現する．また，標準的な publish/subscribe プロトコルである MQTT の他に，AMQP も想定する．しかし，PIQT では MQTT しか対応しておらず，課題の 1 つである．提案手法では，コンポーネントと Wrapper 間を publish/subscribe で繋げているが，Broker による処理遅延が大きく，アプリケーションの要求を満たせない場合，コンポーネントと Wrapper 間は直接接続するなど，別途 Wrapper の実装手法を検討する必要がある．

4. 集約値の更新状況に対するホップ数の推移

3.4 節で述べたように提案したルーティングの性能は集約値の鮮度に依存する．本節では集約値の更新状況によって適切なノードに辿り着くために要するホップ数の影響を調査する．

以下，4.1 節, 4.2 節では提案する Anycast 機構における集約計算遅延が及ぼす影響の評価について述べ，4.3 節ではバスのユースケースにおいての評価について述べる．

4.1 評価方法

本評価では，Dataflow application として，IoT デバイスとそれ以外に 1 つのコンポーネントから構成されるシンプルな Dataflow application を用いる．TV として，本来は CPU 使用率やメモリ使用量などの情報を用いることを想定しているが，集約値の伝搬遅延が処理依頼転送に及ぼす影響を調査するため，3.4 節と同様に単純にコンポーネントの処理可否を 0/1 で表現したものを TV とする．コンポーネントは占有して利用する場合のみを想定し，3.1 節で述べたようにリクエストメッセージによって，処理コンポーネントを選出し，選出されたコンポーネントの処理可否を 0 に変更する．なお，リクエストメッセージの配送には 3.4 節で述べた配送機構を利用する．なお，この 0/1 の変更は Wrapper が担当するため，更新遅延により，新規要求到着時に現在の状態を反映できていない可能性があり，処理可能なノードが存在しない場合には，Wrapper は依頼の再転送を行う．以降の評価では Wrapper による更新遅延は発生しないものとする．以上のような Dataflow application によるリクエストメッセージを用いたコンポーネント占有処理を，全てのコンポーネントが占有されるまでを 1 計測とし，オーバレイ上のノードの総数を 100 として計測を行った．ここでは，処理可能なコンポーネントに到達するまでに要したホップ数，再送回数を計測した．再送回数とは，集約値の更新が間に合わず，処理可能なノードがない Broker に転送された回数を指す．表 3 に各計測で用いたパラメータを示す．メッセージ数は 3.3.3 節で述べた集約値を更新するのにかかる最悪時間 $T(\log n)^2$ の期間に送信するリクエストメッセージ数を示す．該当 topic をもつ Broker 数 ($N(B)$) は，リクエストメッセージを送信する topic| 階層情報 | をもつ Broker 数を示す．Broker1 台あたりの topic| 階層情報 | に該当するコンポーネント数 ($N(C)$) は，実質的に Broker あたりの処理可能な回数となる．計測回数は 50 としホップ数，再送回数の平均，最大，最小を結果に示す．また，パラメータでのメッセージ数の “連続” とは全く更新しない状況を指し，1 の場合は更新する最悪時間で 1 メッセージ発生することを示す．

表 3 パラメータ

	メッセージ数	該当 topic をもつ Broker 数 ($N(B)$)	Broker1 台あたりの コンポーネント数 ($N(C)$)
計測 1	連続 ～ 1	10	1
計測 2	連続, 7	10 ～ 50	1
計測 3	連続, 7	10	1 ～ 5

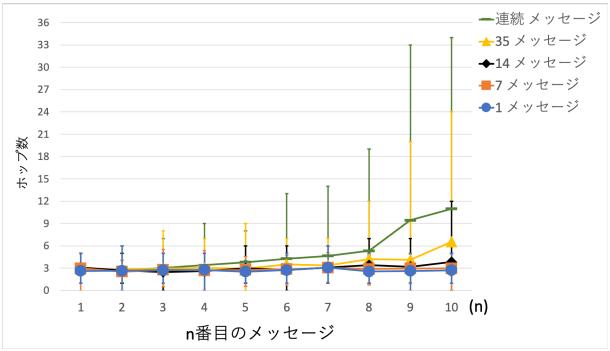

図 11 ホップ数

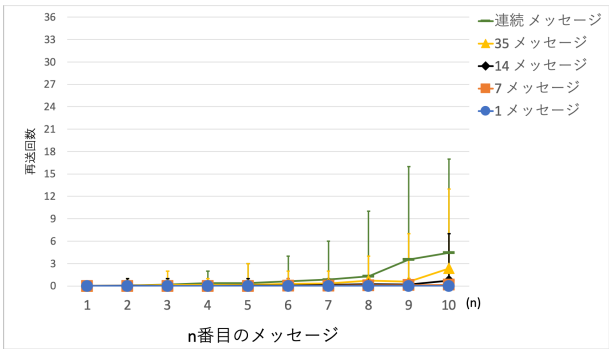

図 12 再送回数

4.2 評価結果

本節では，それぞれのパラメータを踏まえて計測結果と考察を述べる．

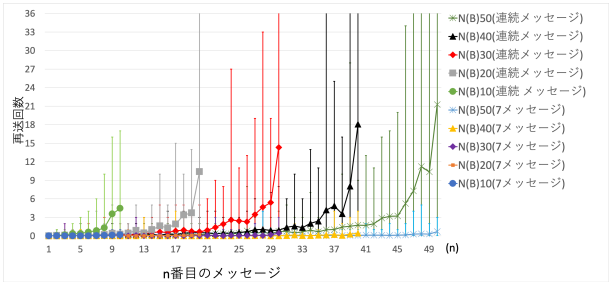

図 13　検索範囲の推移

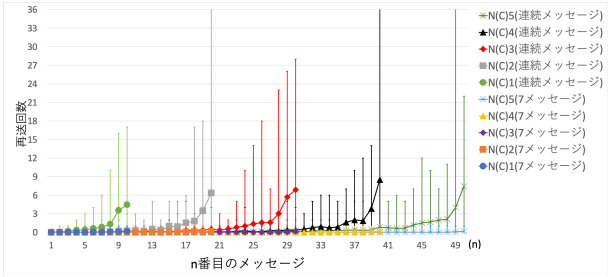

図 14　処理可能な回数の推移

計測 1 の結果のホップ数と再送回数を図 11, 12 に示す．図 11, 12 からメッセージ数 7 のパラメータまでならば，ホップ数，再送回数に大きな影響はないことがわかる．さらに，9, 10 番目のメッセージの結果では，値が急激に増加する傾向にある．これは，最後の方のメッセージの場合，検索範囲内の処理可能なノード数が減少し，集約値のズレが影響しているため他のノードを選出する可能性が高くなり，無駄なルーティングが発生するからである．ホップ数と再送回数は傾向が似ており，以降では再送回数のみのグラフを示す．計測 2, 3 の再送回数のグラフを図 13, 14 に示す．それぞれのメッセージ数 7 の結果では，$N(B)$ が 20 まで，$N(C)$ を増加させても大きな影響はないことがわかる．メッセージ数は両方とも上昇する傾向にあるが，上昇率は検索範囲を大きくした方が高いことがわかる．つまり，集約値の更新状況によるホップ数は検索範囲の大きさとメッセージ頻度に依存する．

4.3　ユースケースに対する有用性

本節では，バスのユースケースにおいて，前節での結果を踏まえて提案手法の有用性を示す．

研究室グループでは神戸市のみなと観光バス株式会社 [23] と連携し，バスに設置した IoT デバイスからデータを取得している．また，データを集める収容局は NTT 西日本が神戸市内で管理している 6 つを用いる．活用する Dataflow application の例として，バスの乗降人数を計測する場合，図 1 のように，活用するコンポーネントは人物検知 → 乗降車判定 → データベースとする．コンポーネントを設置するネットワーク階層は人物検知と乗降車判定を Edge とし，データベースを Cloud1 とする．評価では，

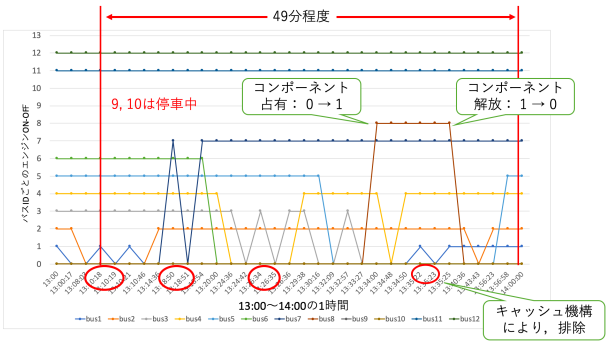

図 15　バスのエンジンをかけるタイミング

IoT デバイスを載せて運行していた 12 台のバスから取得した 2019/01/24 のデータを用いる．この場合，Edge, Cloud 部分は各 Device に配置した 12 台のバスに Broker を 1 台ずつ設置する．6 つの収容局には冗長化や負荷分散を目的にに Broker を 2 台用意し，それらを 1 つの Edge に設置する．Cloud には 1 つのネットワークを用意し，Broker を 10 台設置する．各コンポーネントは均一に Broker に繋がっていることを想定し，オーバレイ上に配置されるノード数は全 Broker 数 32 台，コンポーネント数は 36 個とするとノードの総数は 68 となる．ノード数が 68 の場合，FT の段数は 7 段となる．FT の更新頻度の T はデフォルトで 60 秒に設定されており，k^2T より 49 分で各ノードが保持する全ての集約値が更新される．

図 15 には 2019/01/24 でのバス 12 台に対するエンジンの ON-OFF が切り替わるタイミングを示す．バスの ON-OFF は始発時，昼休憩時，終バスで切り替わり，今回のデータでは一番切り替えが多い昼休憩時である 13:00〜14:00 のデータを扱う．このデータのエンジンのかけ始めの時間から 14 時までが 49 分程度である．この時間帯のエンジンをかける合計回数は 15 回であった．この時間帯では 2 台のバスが走行中で常に 2 個のコンポーネントを占有し，もう 2 台バスが停車し，2 個コンポーネントが余っている状態となる．図 15 の赤丸で示された箇所はエンジンの ON-OFF の間隔が 1 秒程度となっているため，エンジンを掛け直していると考えられる．そこで，Wrapper に確保したコンポーネントの情報を一定時間保持させておくことで，運転手のエンジンの掛け直しによって発生する無駄なメッセージングを抑える．そのため，4 つの赤丸部分を排除することができ，合計回数 11 となる．集約値の更新する最悪時間内に 11 回メッセージが発生する場合，図 11, 12 では，14 メッセージのホップ数と再送回数は最後の結果以外は許容範囲であることがわかる．バスの場合は，余分に 2 つのコンポーネントが配置されているので，4.2 節のようにホップ数が急激に上がる可能性は低いと考えられる．

これらの結果から，提案したコンポーネント間通信手法は，現状のバスのユースケースにおいて，有用であるといえる．

5. まとめと今後の課題

本稿では，Dataflow platform でのコンポーネント間通信における処理ノード選択機構の改善手法を提案した．提案手法では，Pub/Sub 基盤のルーティングでノードが保持する値の集約値を活用し，ノードの負荷状況を考慮したルーティングを可能とした．評価では，集約値の更新頻度に対するメッセージ転送性能の劣化について調査し，想定するユースケースへの影響を明らかにした．また，既存の Dataflow コンポーネントの再利用を容易にするため，Wrapper を用いたコンポーネント間通信機能の分離についても議論した．現状の集約値の更新時間は $T(\log n)^2$ 秒必要であるが，最悪 $2T$ 秒に短縮した FT の更新手法が文献 [14] では提案されており，よりアプリケーションの幅が広がることが期待される．今後の課題として，集約すべき情報の明確化や，Wrapper の実装を行う予定である．

謝辞

本研究開発の一部は JSPS 科研費 17K00143 の助成を受けたものである．

参考文献

[1] “Cisco Visual Networking Index: Forecast and Trends, 2017-2022”. 参照元: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html (アクセス日 2019-04-25)

[2] 飯田 勝吉, “エッジコンピューティング研究開発の現状と今後の課題,” IEEE Access, vol. 117, no. 187 ,pp. 25-30, Aug.2017.

[3] D. Sabella, et al., “Mobile-Edge Computing Architecture: The Role of MEC in the Internet of Things,” IEEE Con- sumer Electronics Mag., vol. 5, no. 4, pp. 84 - 91, Oct.2016.

[4] “ETSI - Multi-access Edge Computing”. 参照元: https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing (アクセス日 2019-04-25)

[5] F.Bonomi, R.Milito, J.Zhu, and S.Addepalli, Fog Computing and its Role in the Internet of Things,“ ACM Workshop on Mobile Cloud Computing, pp. 1316, Aug.2012.

[6] J.Flinn, and M.Satyanarayanan, “Energy-Aware Adaptation for Mobile Applications,” ACM Symp. Operating Systems Principles, pp. 4863, Dec.1999.

[7] Azure IoT Solution Accelerator. 参照元: https://azure.microsoft.com/ja-jp/features/iot-accelerators/ (アクセス日 2019-04-25)

[8] Google Cloud IoT Edge 参照元: https://cloud.google.com/iot-edge/ (アクセス日 2019-04-25)

[9] Y.Teranishi, et al., “Dynamic Data Flow Processing in Edge Computing Environments,” IEEE 41st Annual Computer Software and Applications Conference, Jul.2017.

[10] S. Ishihara et al.,“A Dataflow Application Deployment Strategy for Hierarchical Networks,” IEEE Computer Society Signature Conference on Computers, Software and Applications, 2019 (accepted)

[11] OpenStack 参照元: https://www.openstack.org/ (アクセス日 2019-04-25)

[12] Y.Teranishi, et al., “Scalable and Locality - Aware Distributed Topic - based Pub/Sub Messaging for IoT,” Proc. of IEEE GLOBECOM Dec.2015.

[13] K.Abe, Y.Teranishi, “Suzaku: a Churn Resilient and Lookup-Efficient Key-Order Preserving Structured Overlay Network,” IEICE TRANSACTIONS on Communications, Mar.2019

[14] 安倍 広多，“構造化オーバレイネットワークを用いた条件付きマルチキャストの提案,” DICOMO2019 (採択済み)

[15] T.Schutt, F.Schintke, A.Reinefeld, “Range queries on structured overlay net- works,” Computer Communications, vol.31, no.2, pp.280 - 291, 2008.

[16] MQTT 参照元: http://mqtt.org/ (アクセス日 2019-04-25)

[17] Moquette 参照元: https://github.com/moquette-io/moquette (アクセス日 2019-04-25)

[18] 吉田幹ほか. “マルチオーバレイと分散エージェントの機構を統合した P2P プラットフォーム PIAX,” 情報処理学会論文誌, Vol.49, No.1, pp. 402 - 413, Jan.2008

[19] MQTT Version 5.0 - Name - Oasis 参照元: http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (アクセス日 2019-04-25)

[20] Apache Flink Blog: RSS feed 参照元: https://flink.apache.org/ (アクセス日 2019-05-06)

[21] Node-RED 参照元: https://nodered.org/ (アクセス日 2019-05-06)

[22] Fluentd — Open Source Data Collector — Unified Logging Layer 参照元: https://www.fluentd.org/ (アクセス日 2019-05-06)

[23] 神戸みなと観光バス参照元: http://www.kobe-minato.co.jp/index.html (アクセス日 2019-04-25)