

修士論文

2020 年度

IoT 相互運用性を考慮した透過型分散  
ネットワーク環境の提案と実装

森島遼

(学籍番号 : 81920014)

指導教員 教授 西 宏章

2021 年 3 月

慶應義塾大学大学院理工学研究科  
開放環境科学専攻

# 論文要旨

スマートコミュニティにおけるアプリケーションは IoT センサを活用することで、リモートヘルスケアサービス、自動運転、スマート農業など多分野にわたり、その許容遅延、計算コスト、匿名性、通信データ量は様々である。そのため、スマートコミュニティの通信インフラはクラウドや計算サーバだけでなくネットワーク途中有るエッジ、フォグと呼ばれる領域を有効活用して適材適所に処理をする。アプリケーションに接続する IoT のネットワークには数多くのプロトコルが存在し、異なる仕様の IoT デバイスが連携を取ることが難しいため、ノードには相互にプロトコルを変換する仕組みが求められる。また、エッジ、フォグ領域にあるノードの処理能力は様々であり、アプリケーションの要求を満たすにはノード間で負荷を分散しあう機構がなければならない。しかしながら、一般に IoT デバイスは小型で計算能力が制約されており、数も多いため、ネットワークの再設定や機能のアップデートは最小限にする必要がある。IoT デバイスに一切変更を加えることなく機能追加が行える性質をネットワーク透過性(Network Transparency)と呼ぶ。具体的には、IoT デバイスから見て負荷分散やプロトコルの変換といったネットワーク上の様々な変化が存在しないように見え、また、自身に何ら更新を与える必要がない環境のことを言う。このような理由から、スマートコミュニティの通信インフラはネットワーク透過性を持つことが望ましい。

ノードが自分以外のノードらとコネクションを確立し情報をやり取りすると、コネクション数が多く通信コストが大きくなる。また、ノード数が増加するとシステムがスケールしにくいという問題がある。本研究ではネットワーク透過性を持つディストリビュータと呼ぶノードを使用する。ディストリビュータは既存のインフラを変更することなくネットワークを流れるデータを監視できるという特性を生かし、通信経路を流れるパケットに情報をピギーバックすることで他のディストリビュータとコネクションを確立することなく相互に情報を伝達する。また、ディストリビュータ間でリクエスト処理位置を変更することで負荷分散時でも IoT デバイスは通信を継続することができる。さらに、ディストリビュータは通信途中で様々なプロトコルを変換し、エンドデバイスに一切変更を加えることなく IoT の相互運用性を確保する。

ピギーバックの機能やアプリケーションプロトコルの変換を評価するために、汎用のベアボーン型 PC を用いて実験を行った。アプリケーションプロトコルの変換実験では、IoT デバイスに変更を加えることなくディストリビュータ上でプロトコルを変換することができた。ピギーバックを用いた実験では、ディストリビュータがリクエスト処理位置を柔軟に変更できたこと、ピギーバックが占有帯域と処理遅延に与える影響が非常に小さかったことを確認した。また、ディストリビュータが有するネットワーク透過性によって、IoT デバイスは負荷分散時でも通信を再接続することなくセンサデータを送信することができた。以上よりネットワーク透過性を持ち通信経路上のパケットにピギーバックをするノードを用いてスマートコミュニティの通信インフラを構築することで、IoT 相互運用性を確保しつつアプリケーションの負荷分散を柔軟に行うことができる事を証明した。

# Abstract

## Transparent distributed network environment considering IoT interoperability

The smart grid, which was proposed in the 2000s to modernize the power grid by applying ICT, has gained attention. Efforts to incorporate ICT into infrastructure have been promoted in various social services such as government, medical, and transportation services, and cities equipped with such social infrastructure are called smart communities. Applications in the smart community utilize IoT and cover a wide range of fields such as remote healthcare services and autonomous driving. These applications have different allowable delays, calculation costs, required anonymity, and amount of communication data. Therefore, the network infrastructure of the smart community depends not only on the cloud but also on the edge and fog nodes in order to process the application in the suitable place.

There have been developed many messaging protocols for IoT network and it causes difficulty to share sensor data between IoT devices with different specifications. Edge and fog nodes are required to have a mechanism for converting IoT protocols to each other. In addition, the processing power of the nodes in the edge and fog areas varies, and in order to meet the requirements of the application, there must be a mechanism to balance the load among the nodes. However, it is necessary to minimize network reconfiguration and software updates because general IoT devices are constrained. The solution is that the edge and fog nodes provide for so called network-transparency. Specifically, it refers to an environment in which various changes on the network such as load distribution and protocol conversion do not seem to exist from the IoT devices, and there is no need to modify existing network configuration. It is desirable that the network infrastructure of the smart community has network-transparency.

The conventional load distribution methods in the smart community propose algorithms to minimize the communication between nodes for less transmission cost, but the end device must reconnect to other nodes during load distribution because the nodes in the edge and fog areas do not have network-transparency. In addition, it is necessary to modify the network settings of the existing infrastructure in order to introduce the node into the city.

This research proposes a node called a distributor, which has network-transparency and can monitor the traffic between the IoT device and the cloud. The communication with other distributors is achieved by piggybacking the information to the packet flowing through the communication path. By nature of the network-transparency of the distributors, the IoT device can continue communication during the load balancing. In addition, the distributor converts various IoT protocols in the middle of the communication to ensure IoT interoperability without making any changes to the IoT devices.

# 目 次

<b>1 研究背景</b>	<b>1</b>
1.1 スマートコミュニティとは	1
1.2 スマートコミュニティのインフラストラクチャとサービス	2
1.3 IoT を活用したスマートコミュニティの課題	3
<b>2 関連研究</b>	<b>4</b>
2.1 エッジ, フォグ領域における負荷分散手法	4
2.2 アプリケーションプロトコルの変換手法	6
2.3 Authorized Stream Contents Analysis (ASCA)	8
2.4 ネットワーク透過なストリーム解析ソフトウェア	9
<b>3 提案</b>	<b>11</b>
3.1 概要	11
3.2 ピギーバックによる情報伝達と負荷分散	12
3.3 ネットワーク透過なプロトコル変換	14
<b>4 実験</b>	<b>17</b>
4.1 環境	17
4.2 アプリケーションプロトコルの変換	18
4.3 異なるユーザ設定におけるリクエスト処理位置の変更	19
4.4 リクエスト処理位置の変更による負荷分散の性能比較	20
4.5 通信の一貫性を保ったマイグレーションによる負荷分散	20
<b>5 結果</b>	<b>21</b>
5.1 アプリケーションプロトコルの変換	21
5.2 異なるユーザ設定におけるリクエスト処理位置の変更	23
5.3 リクエスト処理位置の変更による負荷分散の性能比較	26
5.4 通信の一貫性を保ったマイグレーションによる負荷分散	28
<b>6 結論と将来の展望</b>	<b>30</b>
<b>7 参考文献</b>	<b>31</b>

# 図 目 次

図 1. スマートコミュニティのインフラとサービス .....	1
図 2. 様々なネットワーク階層に配置されたアプリケーション群 .....	2
図 3. MtLDF のアーキテクチャ .....	4
図 4. MtLDF のアルゴリズム .....	4
図 5. 各ノードが実行するアルゴリズム .....	5
図 6. Fog Message .....	6
図 7. 共通プロトコルを通す変換手法 .....	6
図 8. プロトコル変換時の中間形式 .....	7
図 9. JSON を用いた中間形式表現 .....	7
図 10. 透明アドオンによるネットワーク途中でのサービス提供 .....	8
図 11. DooR を構成する主なモジュール .....	9
図 12. 想定環境の概要図 .....	11
図 13. ディストリビュータがピギーバックを行い相互に情報伝達するイメージ .....	12
図 14. ピギーバックのデータフォーマット例 計 20 byte .....	13
図 15. 上位のディストリビュータから下位のディストリビュータへのマイグレーション .....	14
図 16. サブスクライブに対するディストリビュータの動作 .....	15
図 17. パブリッシュまたは PUT に対するディストリビュータの動作 .....	15
図 18. GET に対するディストリビュータの動作 .....	16
図 19. 実験環境の様子 .....	17
図 20. 各フィールドの対応関係 .....	18
図 21. ピギーバックによる負荷分散実験のデバイス構成 .....	19
図 22. CoAP による PUT の詳細 .....	21
図 23. 変換された MQTT パブリッシュ .....	22
図 24. 変換された XMPP パブリッシュ .....	22
図 25. X-Header を用いて変換された SMTP .....	23
図 26. 3 台のディストリビュータの負荷を平滑化する場合の負荷の推移 .....	24
図 27. ディストリビュータ A ができるだけ多くのリクエストを処理した場合の負荷の推移 .....	24
図 28. 負荷分散時の拡大図 .....	25
図 29. ピギーバックする頻度と占有帯域の関係 .....	26
図 30. ピギーバックを用いない負荷分散による負荷の推移 .....	27
図 31. マイグレーションによる負荷の推移 .....	28
図 32. 各ディストリビュータが処理したパケットの sequence 番号 .....	29

## 表 目 次

表 1. 実験で用いたデバイスの性能詳細 .....	17
表 2. パーストリクエストの詳細.....	19
表 3. ピギーバックに関する処理の遅延.....	26
表 4. 負荷分散に関する性能比較 .....	27

# 1 研究背景

## 1.1 スマートコミュニティとは

2000年代初頭から、電力網の管理と運営にICTを利用することで電力網の高機能化、統合化を目的とするスマートグリッドが提案されてきた。スマートグリッドは電力需要のオンライン管理やダイナミックプライシングによって、より積極的に発電、蓄電、需要制御を実行するシステムの構築を可能にする。信頼性が高く、効率的なシステムによって電力網の近代化を図るスマートグリッドは世間から注目されるようになった。

このようにスマートグリッドは電力網にICTを織り交ぜることでインフラを高度化する取り組みであるが、この流れは他のインフラにも適用されてきた。行政サービスや医療サービス、運輸サービスなどの様々な社会サービスにICTを導入することでシステムを高度化する取り組みが進められており、電子政府や電子カルテ、物資のリアルタイムトラッキングなどのサービスが実現している(図1)。地域社会の基盤となる社会インフラにICTを活用することで高度なインフラを構築した都市や自治体をその規模に応じてスマートシティ、スマートタウン、スマートヴィレッジなどと呼ぶ。スマートコミュニティとは、シティやタウンなどの地域の規模ではなく、人やインフラも含めた地域社会全体の取り組みに注目した呼称である[1]。

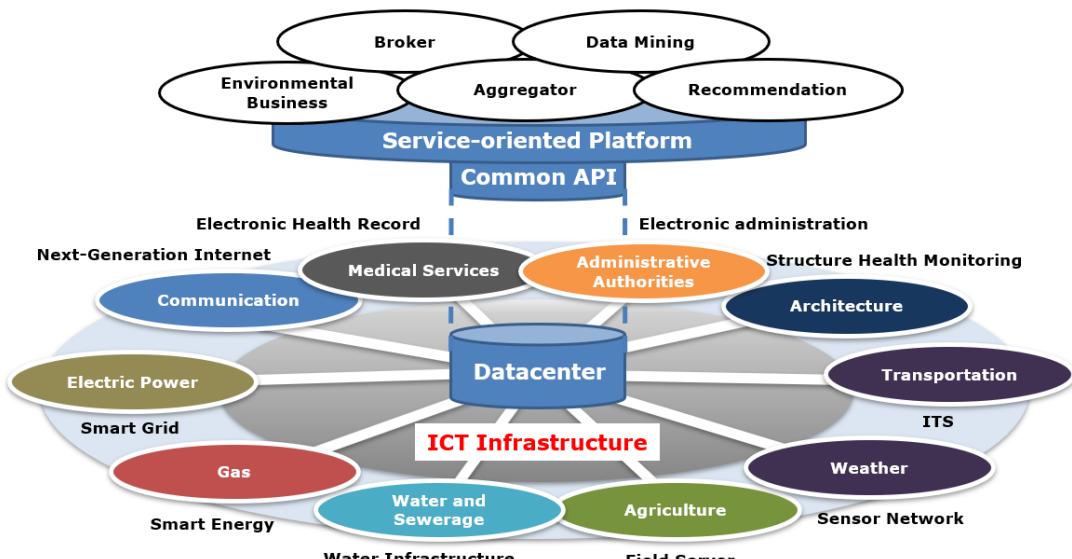


図1. スマートコミュニティのインフラとサービス

## 1.2 スマートコミュニティのインフラストラクチャとサービス

スマートコミュニティにおけるアプリケーションは、スマートグリッドによる電力網制御やリモートヘルスケアサービス、自動運転、スマート農業など多分野にわたり、その許容遅延、計算コスト、匿名性、通信データ量は様々である。例えば自動走行システムで求められるリアルタイム制御では、自動車は取り付けられたセンサが取得する走行状態や交通状態などの情報をサーバへ送信し、サーバは他の自動車の走行状態を考慮したうえで次なる指令を算出、送信する。指令を受信した自動車は実際に加減速、操舵などの制御を行う。自動車とサーバ間の通信データ量と計算コストはデータセンタが必要なほど大きくはないが、処理遅延は 5ms-30ms でなければならぬ[2]。別のアプリケーションとして、地域住民の購買データを収集し、得られたデータを元に住民の趣向を考慮した商品リコメンドを行うサービスを考える。商品のリコメンド情報を提供するのに低遅延を要求されることはないが、購買データは個人の特定につながる可能性があることからデータには匿名化処理を施すことが求められるうえ、機械学習モデルで推論を行うとすればアプリケーションの計算コストは大きい。

このように、スマートコミュニティで想定されるアプリケーションが要求する特徴は多様であるため、クラウドや計算サーバだけでなくネットワーク途中にあるエッジ、フォグと呼ばれる領域を有効活用し、適材適所に処理をする仕組みが必要である。図 2 は許容遅延や匿名性などの要求事項が異なるアプリケーションを、計算能力やネットワーク遅延を考慮してエッジ、フォグ、クラウドといったそれぞれのネットワーク階層に配置した図である。クラウドでは例えば匿名化された、広範囲な地域のデータと高い計算能力を用いてビッグデータの解析や電力市場サービス、災害アラートなど、個人よりは都市全体を対象としたサービスが提供される。対してエッジ、フォグ領域では局地的なデータを用いてヘルスケアや機械制御など、より個人や地域性、遅延を考慮したサービスが提供される。また、ある地域住民の情報を他の地域にむやみに流さない、すなわち地域情報のカプセル化をすべく、匿名化処理などが行われる。スマートコミュニティにおけるサービス提供では、適材適所を配慮した情報処理を行うことが重要であり、その場所を複数提供する通信インフラが求められる。

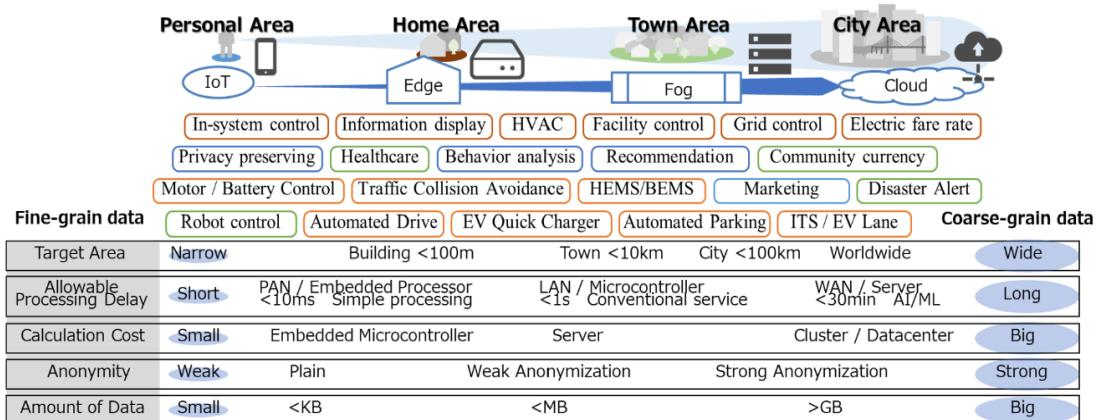


図 2. 様々なネットワーク階層に配置されたアプリケーション群

図 2 に示すようなスマートコミュニティのサービスを実現するには、環境情報を取得するセンサや、機器を物理的に制御するアクチュエータなどの IoT デバイスが必要不可欠である。IoT デバイスは大型で複雑な機能を持った機器から小型でバッテリを持たない機器まで、その種類は多様である。これは一般に IoT デバイスの仕様を決定する要因は、アプリケーションが要求する計算コスト、通信頻度、さらには取り付け可能な面積や重量といった物理的制約など様々存在するからである。

HTTP という1つの標準プロトコルで運用が可能な従来のウェブと異なり、IoT のネットワークは ‘十分な計算能力がない’、‘バッテリの消費を抑えるために通信頻度を低くしなければならない’などの多様な要求に応える必要があるため、数多くのメッセージングプロトコルが提案されてきた [2], [3]。例えば AMQP[4]は高度なメッセージング機能のある信頼性の高いプロトコルである。また、MQTT[5]や CoAP[6]は軽量、省電力なため計算能力が非力な機器にも適している。XMPP[7]は MQTT などと比べて多くの計算リソースを必要とするが、XML データフォーマットを使用してインスタントメッセージング (IM) が可能である。IoT を利用したシステムやサービスでは、このような IoT 機器が持つ制約やアプリケーションの仕様によって異なるメッセージングプロトコルを使い分けているのが現状である。

### 1.3 IoT を活用したスマートコミュニティの課題

スマートコミュニティにおけるアプリケーションはエッジやフォグなど、IoT デバイスとクラウドの中間にあるノードを活用するが、クライアントのリクエストは時間的、地域的な局所性があり、個々のリクエストに対する計算コストはアプリケーションによって異なる。また、中間ノードは商業施設のゲートウェイなどクライアントに近いところでセンサデータを集約するノードや、GPU などのアクセラレータを持ち機械学習モデルの推論を実行するノードなど、場所によって異なる計算能力を有している。そのため中間ネットワークは各アプリケーションがノードに与える様々な負荷とノードの処理能力を考慮しながら負荷を分散させなければならない。

ノードは様々な IoT プロトコルに対応しなければ柔軟に負荷を分散させることができない。プロトコルごとにアプリケーションを管理すると、あるプロトコルで運用するアプリケーションの負荷が増加した場合に仕事を他のノードに分担させることができないからである。また、サービスによって異なる IoT プロトコルを採用すると、システムの相互運用性が乏しく異なる仕様の IoT デバイスが連携を取ることが難しい。異なるプロトコルを使用している機器間で通信を行うにはプロトコルを変換する必要があるが、センサなどの小型デバイスは低消費電力、低成本、シンプルであることが求められるため、エンドデバイスにおける複数プロトコルへの対応やプロトコルの変換処理は困難である。したがって、IoT の相互運用性を確保するにはエッジやフォグなどにおいて通信プロトコルを相互に変換する必要がある。

このように IoT デバイスに一切変更を加えることなく新しい通信プロトコルへの対応などの機能追加が行える性質をネットワーク透過性 (Network Transparency) と呼ぶ。具体的には、IoT デバイスから見てサービス提供場所の変更やプロトコルの変換といったネットワーク上の様々な変化が一切存在しないように見え、また自身に何ら変更を加える必要がない環境のことを言う。このような理由から、スマートコミュニティの通信インフラはネットワーク透過性を持つことが望ましい。

## 2 関連研究

### 2.1 エッジ, フォグ領域における負荷分散手法

フォグやエッジのネットワークはノード数が多く、加えてノードの処理能力やリクエストに対する応答遅延などを考慮しなければならないことから様々な負荷分散手法が提案されている。スマートコミュニティのようなインフラを想定したフォグ領域における負荷分散手法として Multi-tenant Load Distribution Algorithm for Fog Environments (MtLDF)[8]が挙げられる(図 3)。

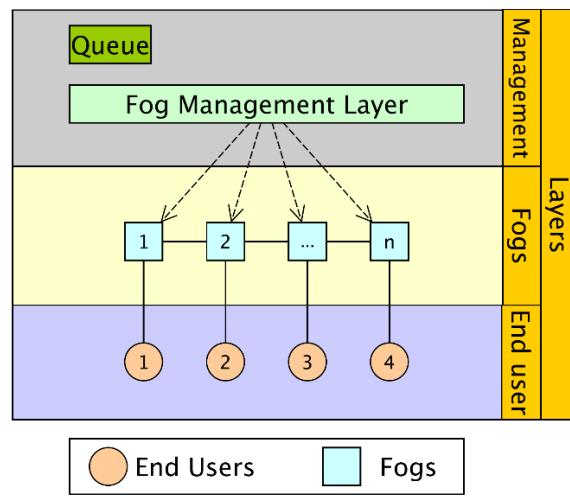


図 3. MtLDF のアーキテクチャ[8]

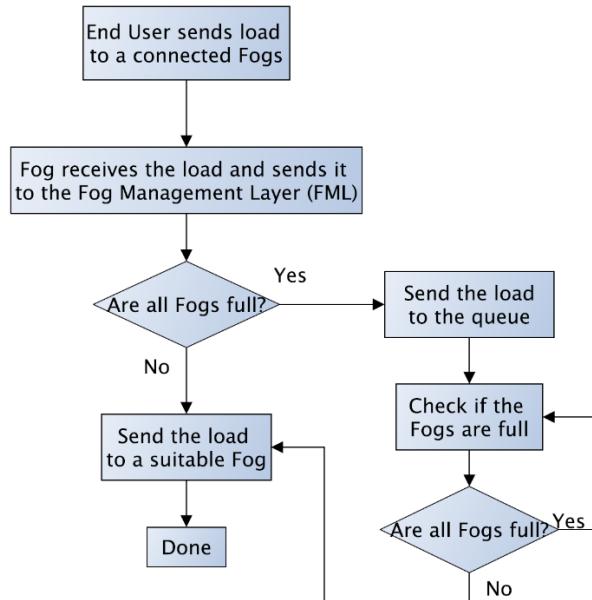


図 4. MtLDF のアルゴリズム[8]

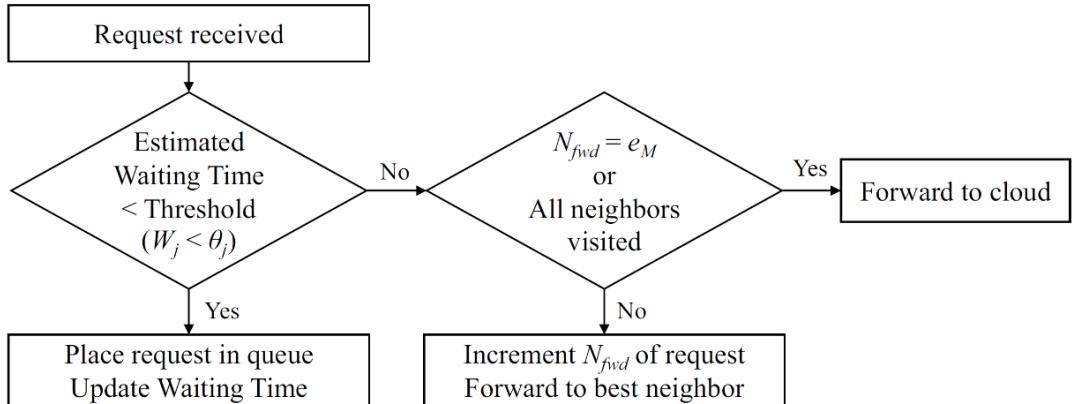


図 5. 各ノードが実行するアルゴリズム[9]

この手法ではマスターノードが各ノードの負荷情報を収集し、クライアントからのリクエストに対する応答遅延などを考慮し最適なノードに仕事を割り振ることで負荷を分散する(図 4).しかし、全ノードの負荷情報を収集、比較する手法はノード数の増加に伴い計算コストが大きくなるため、大きなネットワークでは遅延が増大するという問題がある。

マスターノードを用いずに各ノードが相互に連携を取ることによって負荷を分散する手法も提案されている。Yousefpour らが提案する手法[9]は各ノードが自身の負荷状態を見てリクエストを処理するか判断し、できないと判断した場合は負荷の小さい隣接ノードにリクエストをフォワードする。もしすべてのノードに到達した、最大フォワード数に達したなど、フォグ領域で処理ができなかった場合はクラウドにあるサーバにフォワードする(図 5).

[10]-[12]も同様のアルゴリズムを採用しているが、隣接ノードを選択する際に Power of Two Choices と呼ばれる選択モデルを実行するなどランダム性を入れることで比較する隣接ノード数を減らし、各ノードが隣接ノードの負荷を管理するコストを小さくしている。MtLDF のようなマスターノードが各ノードの負荷情報を収集する一元的な手法と比べて負荷を制御するための通信コストや計算コストは小さくなるが、負荷分散の精度は低下する。特にバーストリクエストのような局所的な負荷に対する応答速度が小さく、アプリケーションの QoS が低下する。

以上の手法ではすべてのタスクが要求するコンピューティングリソース量は同一であると単純化しているが、実アプリケーションはメモリインテンシブなタスクや CPU の計算速度が重視されるタスクなど様々である。Andreas らはタスクを別なノードに割り当てる際にそのタスクのメモリ使用量やデータサイズなどを考慮する手法を提案している[13]. 図 6 は Fog Message と呼ばれるノード同士が交換するメッセージを示しており、タスクの内容(Data, Execution script)とタスクの実行に必要な処理の複雑さなど(Metadata)が記述されている。しかしながら各ノードの負荷情報はマスターノードに収集されるため、MtLDF と同様の問題が生じる。

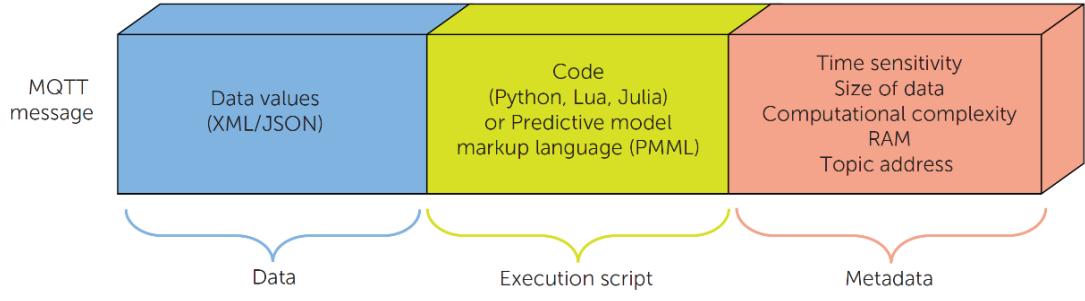


図 6. Fog Message[13]

また、本節で列举した関連手法はフォグ領域にあるノードが IoT デバイスから見てネットワーク透過ではないため、新しいノードを追加する場合や IoT デバイスが別なノードに処理を依頼する場合は IoT デバイスを再接続させなければならない。

## 2.2 アプリケーションプロトコルの変換手法

シンプルなアプリケーションプロトコルの変換手法は、あるプロトコルから別なプロトコルへ直接変換することである。例えば[14]は MQTT と XMPP について、[15]は MQTT と CoAP について相互変換を行っている。このようにプロトコル同士を直接変換する手法は対応するプロトコル数を  $N$  とすると  $N(N - 1)$  通りの処理を実装しなければならず、扱うプロトコルが多い場合は現実的な手法ではない。この問題を回避するため、Bouloukakis らは一度プロトコルを共通プロトコルに変換し、共通プロトコルを目的のプロトコルに変換するという手法を用いている[16]。図 7 は MQTT を REST に変換する際に、一度共通プロトコルである CoAP に変換する処理を挟んでいる様子を示している。

共通プロトコルを用いる手法はプロトコル数を  $N$  とすると  $2(N - 1)$  通りの変換処理数となり、プロトコル同士を直接変換する手法と比べて多数のプロトコルに対応することができる。しかし、あるプロトコルが持つ QoS や PING 要求などの機能に共通プロトコルが対応していない場合は正しく変換することができない。また、共通プロトコルを変更する場合は変換処理をすべて実装し直さなければならない。この問題は共通の形式に変換する際に特定のプロトコルに依存しないデータフォーマットに変換することで解決することができる。[17]はプロトコルを一度図 8 に示すような7つのエントリを持つ中間形式に変換する手法を提案している。

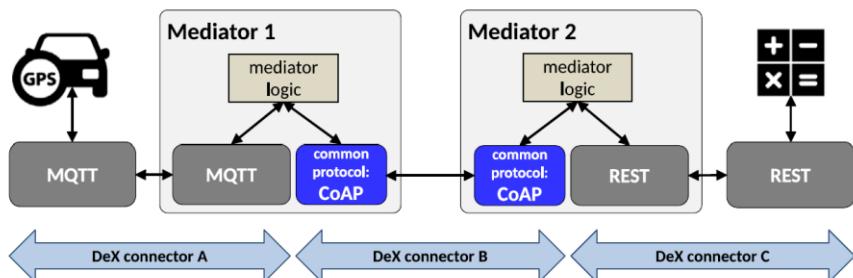


図 7. 共通プロトコルを通す変換手法[16]

BaseContext
-uniqueKey : int -method : string -object : string -query : string -payload : string -payloadFormat : string -exception : string

図 8. プロトコル変換時の中間形式[17]

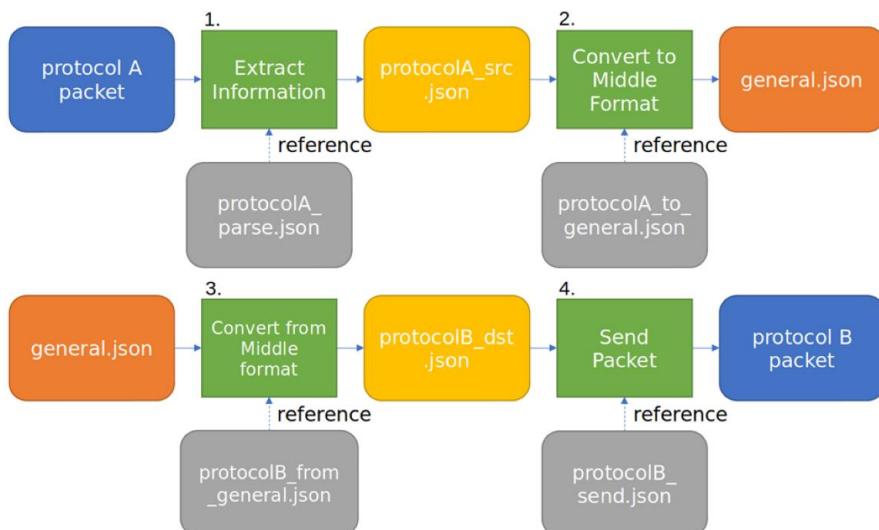


図 9. JSON を用いた中間形式表現[18]

慶應義塾大学西宏章研究室(以下「西研究室」と称する)ではスマートコミュニティのようなマルチテナントな環境を考慮し、プロトコルに依存しない中間形式に変換することに加えてプロトコルを変換する際のルールをユーザが自由に定義できる手法を提案した[18]. この手法では、プロトコル変換時の中間形式や変換する際に参照するルールファイルはすべてJSON形式で表現される。JSONを用いたアプリケーションプロトコルの変換の流れは次の通りである(図 9).

1. Extract Information  
アプリケーションのペイロードから ID やメッセージなどの情報を抽出し各情報を JSON として出力する。
2. Covert to middle format  
抽出した変換元プロトコルの情報からプロトコルに依存しない中間形式にマッピングする。中間形式は JSON で表現される。

### 3. Convert from middle format

中間形式から変換対象のプロトコルへマッピングを行う。変換対象のプロトコル情報も変換元プロトコルと同様に JSON で記述される。

### 4. Send packet

変換対象のプロトコル情報を元にパケットを作成し送信する。

また、ペイロードのパース時やプロトコル情報から中間形式に変換する際にユーザが定義した変換ルールが記述された JSON ファイルを参照する(図 9 灰色枠)。変換ルールではプロトコルの各フィールドをどのような形式(整数型, 文字列型など)で抽出するかを正規表現や分岐処理を用いて定義する。ユーザは自身のアプリケーションの仕様に応じて独自のルールファイルを用いることでプロトコルから必要な要素のみ取り出すことができる。本研究におけるアプリケーションプロトコルの変換は[18]と同様に中間形式を介するアーキテクチャを用いる。

## 2.3 Authorized Stream Contents Analysis (ASCA)

ASCA とは、パケットのヘッダのみならずペイロードを解析することでコンテンツに応じたサービス提供を可能とする技術である[19], [20]。ASCA は例えば HTTPS や MQTT over SSL などの暗号化された通信であってもクラウドと結託して入手した共通鍵を用いて暗号を復号する。gzip 展開や chunk デコードを行いつつ、コンテキストスイッチ機能とコンテキストキャッシュ管理により TCP ストリームを再構築する。また、ストリームの中から必要な情報を正規表現などにより抜き出し、情報抽出や改変を行うことが可能である。

ASCA を実装したエッジコンピューティングノードを用いることでエッジ領域において透明アドオンと呼ぶサービス提供方法を実現することができる[21]。ここで述べた透明とはネットワーク透過性を有することを指す。透明アドオンを活用することで、エンドデバイスの機能拡張やネットワーク設定の変更をせずに、エンドデバイスにない機能をエッジ領域で自由に追加することができる。さらに、追加機能が透明であるがゆえにエンドデバイスはもともと持っていた機能を維持することができる。したがって、ASCA によりエッジ領域においてネットワーク透過なプロトコル変換、データ変換、セキュリティ機能の追加などが実現可能になる。図 10 はゲートウェイやエッジなどの異なるネットワーク階層における透明アドオンにより、ネットワーク途中でサービス提供が行われるイメージ図である。

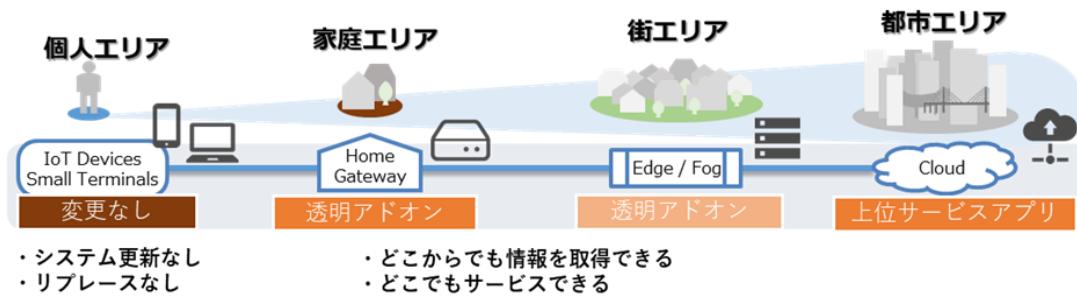


図 10. 透明アドオンによるネットワーク途中でのサービス提供

## 2.4 ネットワーク透過なストリーム解析ソフトウェア

西研究室が開発してきた、DooR と呼ばれるストリーム解析ソフトウェアを紹介する。DooR は通常のルータの動作に加えて ASCA の機能を備えており、ペイロードを抽出、分析し、それをサービス提供に活用する。DooR はエンドデバイスのように通信を確立しパケットを作成することはせず、自身を通過するパケットを監視、解析するため、ネットワーク透過である。DooR は Intel DPDK[22]が提供するライブラリとドライバを用いて C 言語で開発され、Linux OS がインストールされた汎用マシンで動作する。図 11 に DooR の内部構成を示す[23]。

DooR は通信経路上のセッションを特定してサービスを提供するために、フラグメントされたパケットを TCP ストリームに再構築する処理を行う。ここで言うストリームとは送信元から宛先への単方向の通信である。DooR には同じストリームに属するパケットが連續して通過するのではなく、複数の異なるストリームに属するパケットが混在して流れるため、パケットがどのストリームに属するか判別する必要がある。DooR は通過するパケットから 5-tuple(ファイブタプル)と呼ばれる、送信元 IP アドレス、宛先 IP アドレス、送信元ポート、宛先ポート、プロトコル番号からなる 5 つの情報を取り得しストリームを識別する。

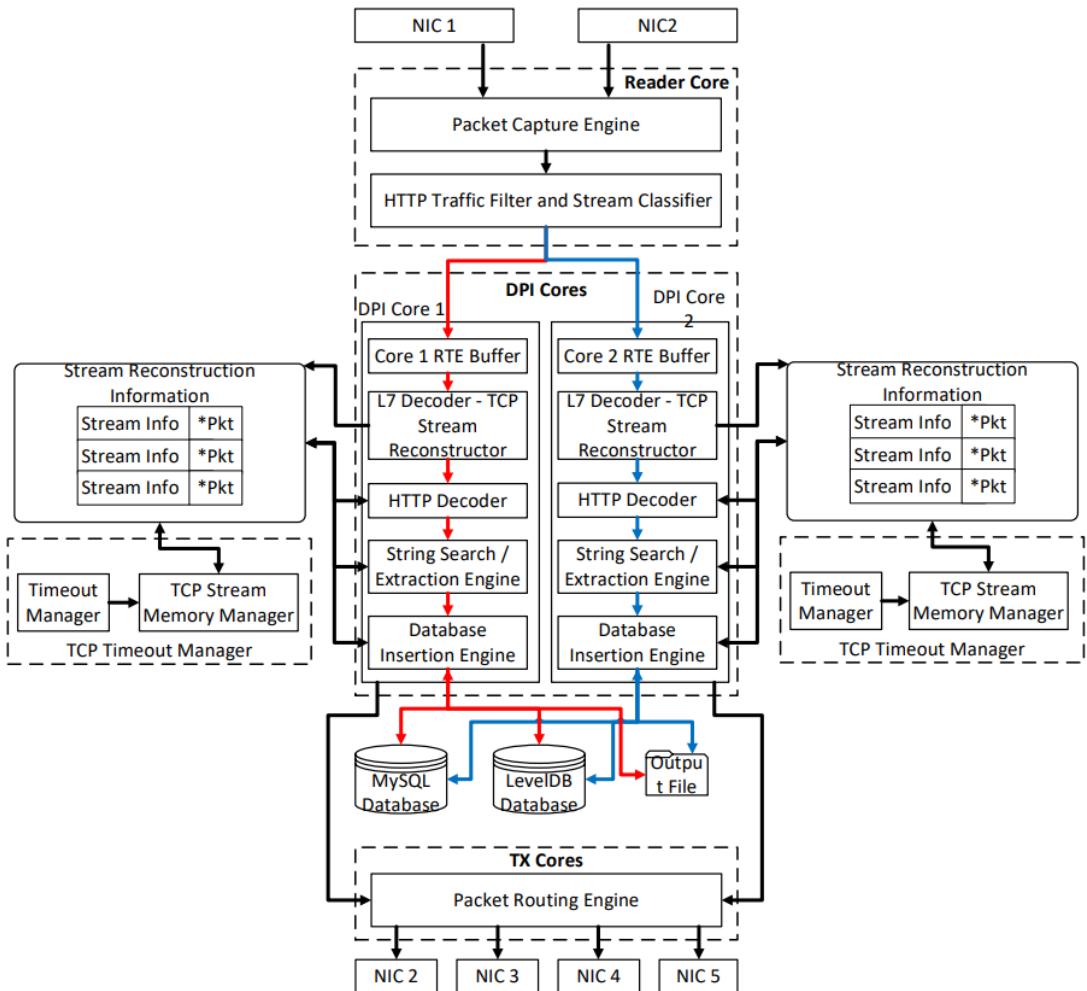


図 11. DooR を構成する主なモジュール[23]

エッジ領域のネットワークにおいてパケットをダンプした場合, そのサイズは数 TB/h に及ぶため, DooR をデプロイしたマシンをエンドデバイス-クラウド間に配置した場合, すべてのパケットペイロードを保存することはストレージ不足やディスク I/O のボトルネックにより不可能である. さらに, 仮にすべてのトランザクションを保存したとしても実際にサービスとして活用される部分は一部である. DooR はすべてのペイロードをデータベースに保存するのではなく, あらかじめ指定した文字列にマッチする情報を探索し, 条件によって処理または保存する.

DooR は文字列マッチングルールがパケットペイロードに部分的にマッチすると, 文字列マッチングのステートをその TCP ストリームに紐づけて保存する. 次にその TCP ストリームに属するパケットを検知した場合, 保存した文字列マッチングのステートを復元する. これにより DooR はパケット単位の文字列マッチングにとどまらず複数のパケットにまたがった文字列マッチング処理を実現する. DooR は文字列検索処理として Mischa Sandberg によって開発された Aho-Corasick アルゴリズム[24]や正規表現マッチングライブラリである Intel Hyperscan[25]を利用する. Intel の評価[26]によると Hyperscan は最大で 160Gbps までスループットがスケールする.

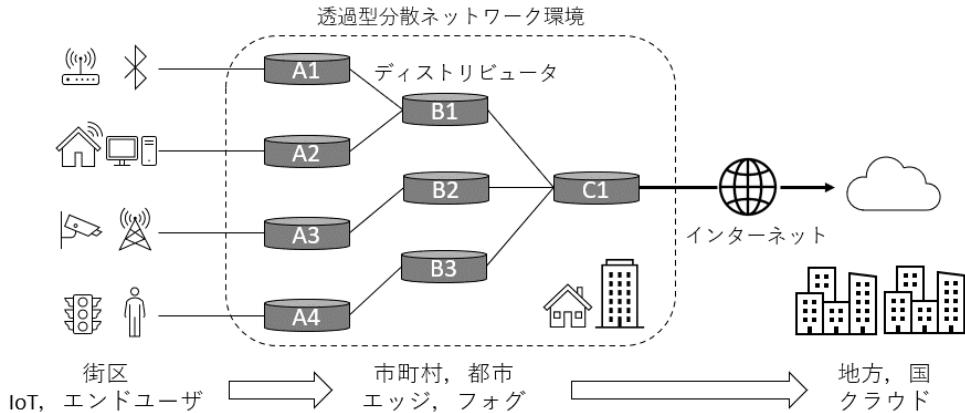


図 12. 想定環境の概要図

### 3 提案

#### 3.1 概要

図 12 に本論文で想定する透過型分散ネットワーク環境の概要を示す。IoT デバイスおよびエンドユーザはクラウドと通信しており、その中間にネットワーク透過性を持った分散ネットワーク環境が位置している。分散ネットワーク環境を構成するノードは、後述するピギーバックによる情報伝達機能と透過的なプロトコル変換機能によって負荷情報や多様な IoT メッセージを発信するノードであることから、本研究ではこれらをディストリビュータと呼ぶ。この環境はスマートコミュニティにおけるプラットフォームとして様々なユーザがディストリビュータ上に匿名化機構やブローカ機能などのアプリケーションを配置することを想定する。

あるディストリビュータに配置されたアプリケーションで行う処理は、通信経路上の他のディストリビュータでも同様の処理を行うことで負荷を分散することができる。IoT デバイスが送信するセンサデータに対し、クラウドに到達する前にエッジ領域でタグ付けや前処理を施すアプリケーションを考える。センサデータが図 12 におけるディストリビュータ A1, B1, C1 を通過すると分かれれば、例えばアプリケーションを A1, B1 に配置し、2 つのディストリビュータのリクエスト処理量を変動させることによってアプリケーションの負荷を分散させることができる。さらに、A1 もしくは B1 のアプリケーションを通信経路上の空いているディストリビュータ (C1) にマイグレーションすることによっても負荷を分散させることができる。

このような負荷分散を実行するには、通信経路上のディストリビュータが相互に負荷情報や他のディストリビュータに対する制御情報を通達しなければならない。このときディストリビュータが自分以外のディストリビュータらと接続を確立し情報をやり取りすると、接続数が多く通信コストが大きくなる。また、ディストリビュータ数が増加すると情報の伝達に時間がかかり、システムがスケールしにくいという問題がある。代わりに、ネットワーク透過性によって既存のインフラを常時監視できるというディストリビュータの特性を生かし、通信経路を流れるパケットに負荷

情報や制御情報をピギーバックすることを考える。ディストリビュータは互いにコネクションを確立する必要がなく、効率よく情報を伝達することができる。

また、ディストリビュータはネットワーク途中でアプリケーションプロトコルの変換を行うことにより異なるプロトコルによるリクエストをまとめて負荷を分散させることができる。その際、TCP通信の一貫性を維持しながら別なプロトコルへの変換、パケットの生成を行うことでエンドデバイスにプロトコル変換処理を気づかせない。ディストリビュータのネットワーク透過性とアプリケーションプロトコルの変換によってIoTデバイスはBluetoothやWi-Fiなどを用いた既存の通信インフラを変更することなく使い続けることができ、IoTデバイスの相互運用性を確保する。

### 3.2 ピギーバックによる情報伝達と負荷分散

ディストリビュータは自身のIDなどのメタ情報や負荷情報をIoTデバイス-クラウド間の通信にピギーバックとして付加することで他のディストリビュータに伝達する。エンドデバイス間通信のアップストリームとダウンストリーム両方を利用して通信経路上のディストリビュータは上流-下流間で情報をやり取りすることができる。付与されたピギーバックは分散ネットワーク環境からインターネットに出る際にエンドデバイス間通信の一貫性を保つために除去される。図13はディストリビュータがエンドデバイス間通信にピギーバックをするイメージ図である。また、ピギーバックに記載するデータの例を図14に示す。この例では20byteほどのデータサイズでディストリビュータのCPU使用率や使用可能なメモリ量、アクセラレータの有無などの情報を伝達することができる。ディストリビュータによるインフラはピギーバックによって交換された負荷情報をアプリケーション作成者(ユーザ)に提供する。ユーザは提供される負荷情報を利用することで各ディストリビュータにおけるアプリケーション負荷を次式のように概算することができる。

$$Load_{app1} = w_1 \times l_1 + w_2 \times l_2 + \cdots + w_n \times l_n$$

$l_n$ はCPU使用率やメモリ使用量などの各負荷のスコアを、 $w_n$ は各負荷に対する重みを表している。ユーザはアプリケーションの特徴に応じて重み $w_n$ を調整することで負荷を表現する。ディストリビュータのネットワークはユーザが定義したアプリケーション負荷の値を元にピギーバックを用いてリクエスト処理位置を変更する。エンドデバイス側を下位のネットワーク、クラウド側を上位のネットワークとすると、例えば上位のディストリビュータは通信経路におけるアップストリームのピギーバックを調べることで下位のディストリビュータのアプリケーション負荷を知ることができます。

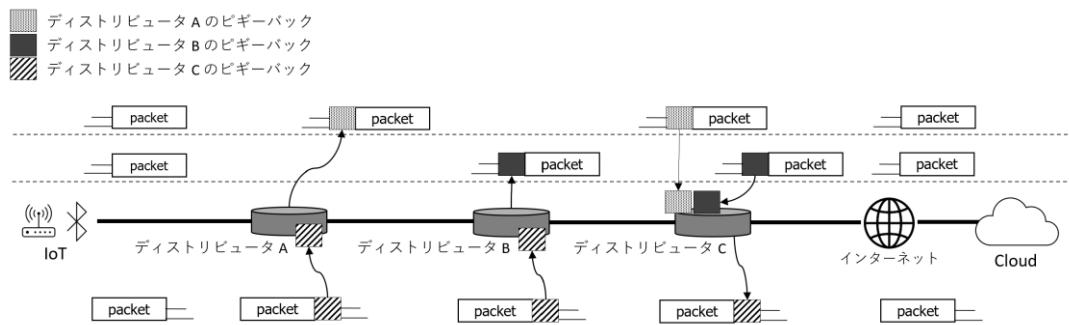


図13. ディストリビュータがピギーバックを行い相互に情報伝達するイメージ

- 4-byte ヘッダ
  - ディストリビュータのメタ情報
  - ID, バージョン
- 4-byte タイムスタンプ
- 2-byte 使用可能なRAMサイズ、ロードアベレージ
- 2-byte リクエスト処理時間の平均値
- 2-byte ディスク I/O, CPU使用率
- 2-byte ネットワーク I/O
- ~2-byte フラグ
  - GPUの有無
  - FPGAの有無
  - DBの有無
  - マイグレーション用のフラグ
  - パケット処理用のフラグ
- 2-byte 追加情報
  - GPUやFPGAの詳細

図 14. ピギーバックのデータフォーマット例 計 20 byte

る。上位のディストリビュータはダウンストリームにアプリケーションのリクエスト処理量を変更する指令をピギーバックし、経路上の他のディストリビュータにリクエストを分散させることができる。

ユーザが経路上に複数のアプリケーションを配置していない場合は、アプリケーションをマイグレーションすることによっても負荷を分散させることができる。稼働中のアプリケーションをネットワーク途中でマイグレーションするときはクライアントの通信が破壊されないようにしなければならない。想定環境のような階層ネットワークにおけるマイグレーションは、1) 下位から上位のディストリビュータへのマイグレーションと、2) 上位から下位のディストリビュータへのマイグレーションの2つに大別することができる。

1) 下位から上位のディストリビュータにアプリケーションをマイグレーションする際は、上位のディストリビュータがマイグレーション中にクライアントが送信するパケットをバッファリングする。マイグレーションが完了したらバッファリングしていたパケットを開放しアプリケーションに受け渡す。上位のディストリビュータはマイグレーション実行と同時にパケットバッファリングを開始すれば通信の一貫性を保つことができ、シンプルな実装で達成される。

2)しかし、上位から下位のディストリビュータにアプリケーションをマイグレーションするときは、マイグレーションを実行する前に下位のディストリビュータがクライアントから送信されるパケットをバッファリングしなければ通信の一貫性を保つことができない。さらに、上位のディストリビュータは下位のディストリビュータがバッファリングする前のパケットをすべてアプリケーションに渡してからマイグレーションを開始する必要があるため、通信中を流れるパケットの中でバッファリングされなかつた最後のパケットを特定できなければならない。

ピギーバックを用いてパケットに印をつけることで通信の一貫性を維持した2)のマイグレーションを実現する(図15)。①上位のディストリビュータはダウンストリームにピギーバックし下位のディストリビュータにマイグレーションを実行することを通知する。②下位のディストリビュータはバッファリングを開始する。また、アップストリームにピギーバックし通信内のバッファリングの境界を上位のディストリビュータに通知する。上位のディストリビュータは通知が付与されたパケットとそれ以前のパケットを処理する。③下位のディストリビュータがバッファリングする前のパケットをすべて処理した上位のディストリビュータはマイグレーションを実行する。マイグレーションが完了したら下位のディストリビュータはバッファ内のパケットを開放しアプリケーションに渡す。

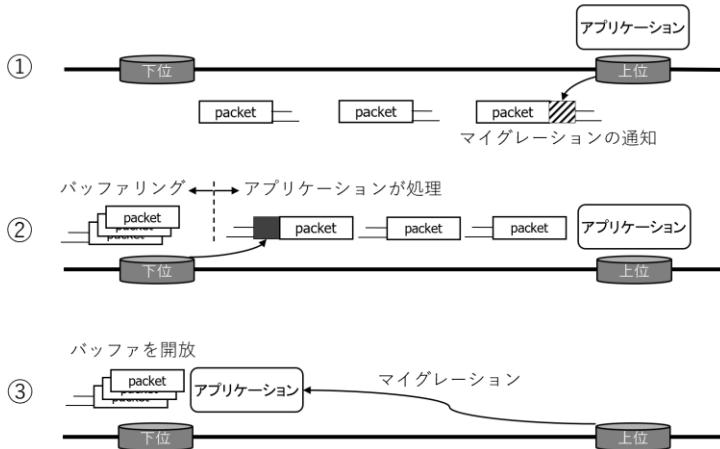


図 15. 上位のディストリビュータから下位のディストリビュータへのマイグレーション

### 3.3 ネットワーク透過なプロトコル変換

ディストリビュータはパブリッシュ/サブスクリープ形式に加え, HTTP 等の REST 形式のプロトコルに対応し相互に変換する。パブリッシュ/サブスクリープ形式の通信では, パブリッシュおよびサブスクリープのメッセージはトピックを宛先として送信される。REST 形式の通信で用いられる URI をトピックとして用いることでパブリッシュ/サブスクリープ形式とデータのやり取りを行う。パブリッシュ, サブスクリープおよび REST の命令に対するディストリビュータの動作を次に示す。

- **サブスクリープ (図 16)**  
サブスクリーバが TCP ストリーム  $s_n$ においてプロトコル  $p_n$ を用いてトピック  $t_n$ にサブスクリープを送信する。ディストリビュータは通過するサブスクリープのパケットを解析しトピック  $t_n$ とプロトコル  $p_n$ および TCP ストリーム  $s_n$ の情報を保存する。TCP ストリーム情報として保存する内容には 5-tuple に加え, sequence 番号や acknowledge 番号の情報が含まれている。TCP ストリーム  $s_n$ に該当するパケットが流れた場合, 常に sequence 番号や acknowledge 番号などの TCP ストリーム情報を更新する。
- **パブリッシュおよび PUT (図 17)**  
パブリッシャはパブリッシュもしくは PUT を用いてトピック  $t_n$ にメッセージ  $m_n$ を送信する。ディストリビュータはトピック  $t_n$ のサブスクリーバが使用したプロトコル  $p_1 \sim p_n$ を用いてメッセージ  $m_n$ のパブリッシュを作成する。サブスクリーバが属する TCP ストリーム  $s_1 \sim s_n$ の sequence 番号と acknowledge 番号を元に連続したパケットに見えるようにパブリッシュパケットを作成しサブスクリーバに送信する。また, ディストリビュータはトピックに紐づけて最新のメッセージを1つだけ保存し, 後述する GET に対応する。
- **GET (図 18)**  
トピック  $t_n$ に対応する最新のメッセージ  $m_n$ を取得し, 同じアプリケーションプロトコルを用いてレスポンスを送信する。GET に応答する場合はサブスクリーバを意識する必要はない。

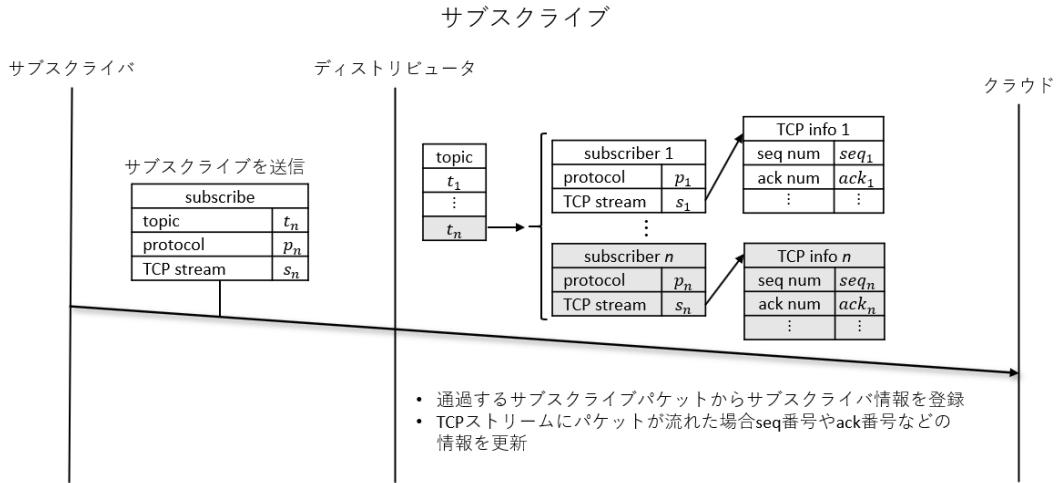


図 16. サブスクリーブに対するディストリビュータの動作

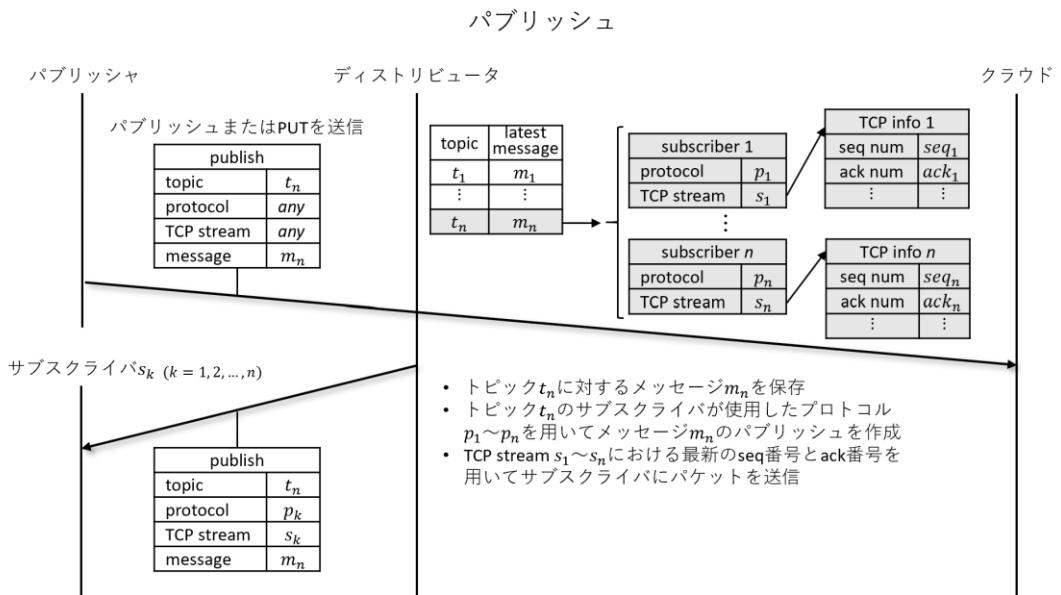


図 17. パブリッシュまたは PUT に対するディストリビュータの動作

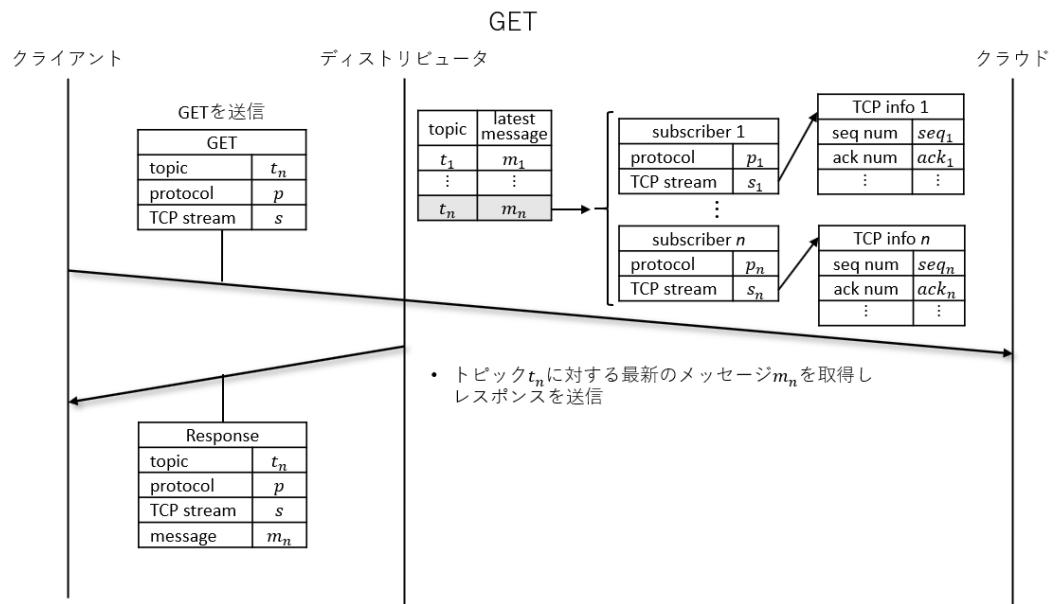


図 18. GET に対するディストリビュータの動作

## 4 実験

### 4.1 環境

実験環境は IoT デバイス、ディストリビュータ、クラウドの 3 つの役割から構成される。ディストリビュータはエッジ領域で使用されることを想定し、小型の汎用ベアボーン PC (Shuttle DH310[27]) を用いて実験を行った。ディストリビュータに DooR をインストールし、ピギーバックやアプリケーションプロトコルの変換などの機能を実装し実験を行った。IoT デバイス、クラウドの役割を持つデバイスにはそれぞれ Intel® NUC kit NUC6i3SYK[28]、Intel® NUC kit NUC8i5BEK[29]を使用した。表 1 に実装に使用したデバイスの性能を示す。図 19 は以上の複数のベアボーン型 PC で構築された実験環境を示している。

表 1. 実験で用いたデバイスの性能詳細

	Shuttle DH310	Intel® NUC kit NUC8i5BEK	Intel® NUC kit NUC6i3SYK
OS	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS	Ubuntu 18.04.4 LTS
CPU	Intel® Core™ i7-8700 Processor (12M Cache, up to 4.60 GHz)	Intel® Core™ i5-8259U Processor (6M Cache, up to 3.80 GHz)	Intel® Core™ i3-6100U Processor (3M Cache, 2.30 GHz)
RAM	32GB	8GB	8GB
Link speed	1 Gbps	1 Gbps	1 Gbps

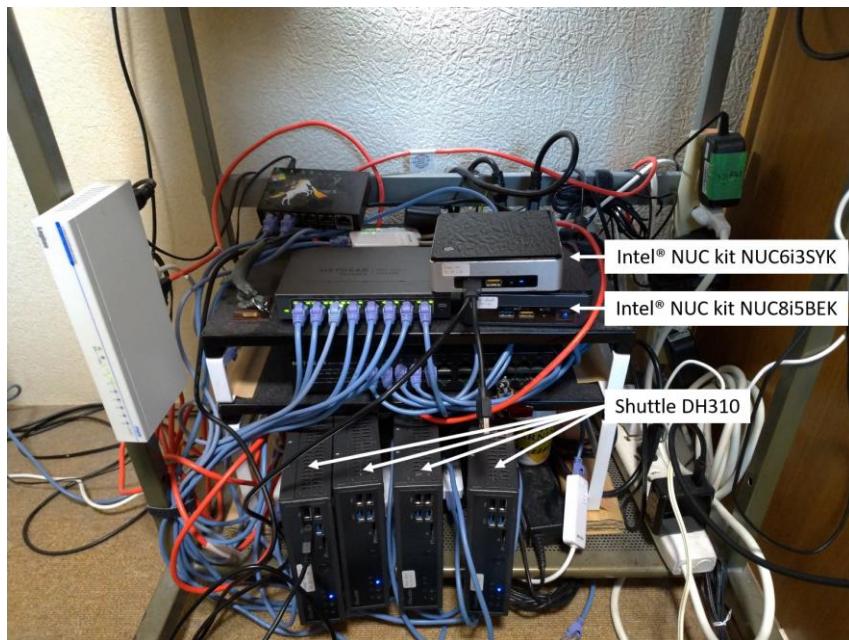


図 19. 実験環境の様子

## 4.2 アプリケーションプロトコルの変換

ディストリビュータにおけるアプリケーションプロトコルの変換を実験するために、ディストリビュータに MQTT, XMPP, CoAP, SMTP 計4つのアプリケーションプロトコルを変換可能なブローカ機能を実装した。パブリッシュ/サブスクライブ形式のプロトコルとして MQTT および XMPP を、REST 形式のプロトコルとして CoAP を選択した。また、e メールによるパブリッシュ/サブスクライブを実現しクライアントの利便性を向上させるために SMTP を加えた。図 20 は MQTT, XMPP, CoAP, SMTP における各フィールドの対応関係を示しており、対応するフィールドには同じ記号がついている。異なる通信形式のプロトコル間でデータ連携することを目指し、QoS などのオプションは考慮しなかった。MQTT のトピック、メッセージはそれぞれ XMPP の node, value, CoAP の URI, data, に対応する。SMTP では、ユーザが自由に記述できることを考慮し Subject(メール件名)などのフィールドには手を付けず、X-Headerを利用した。X-Headerとは任意の値を書き込むことができるヘッダであり、拡張機能やメールサーバのメタ情報などが書かれていることが多い。“X-”から始まる変数名と、変数名に対する値をコロン:で接続して表現する。例えば、“X-MyID: 12345”的ように記述する。実験ではメールヘッダに”X-mqtt\_over\_smtp: true”と記述するとパブリッシュ/サブスクライブ形式として扱われ、X-subscribe, X-publish に指定された値がそれぞれサブスクライブ、パブリッシュ時のトピックとして解釈されるように実装した。

実験では IoT デバイス、ディストリビュータ、クラウドを有線で連続に接続した。IoT デバイスはクラウドに対し MQTT や XMPP を用いてサブスクライブ、パブリッシュを行い、ディストリビュータが IoT デバイスに対しネットワーク透過的にプロトコル変換を実行できるかを確認した。また、TCP ストリームやアプリケーションプロトコルのステート管理に必要なメモリ使用量を調べた。

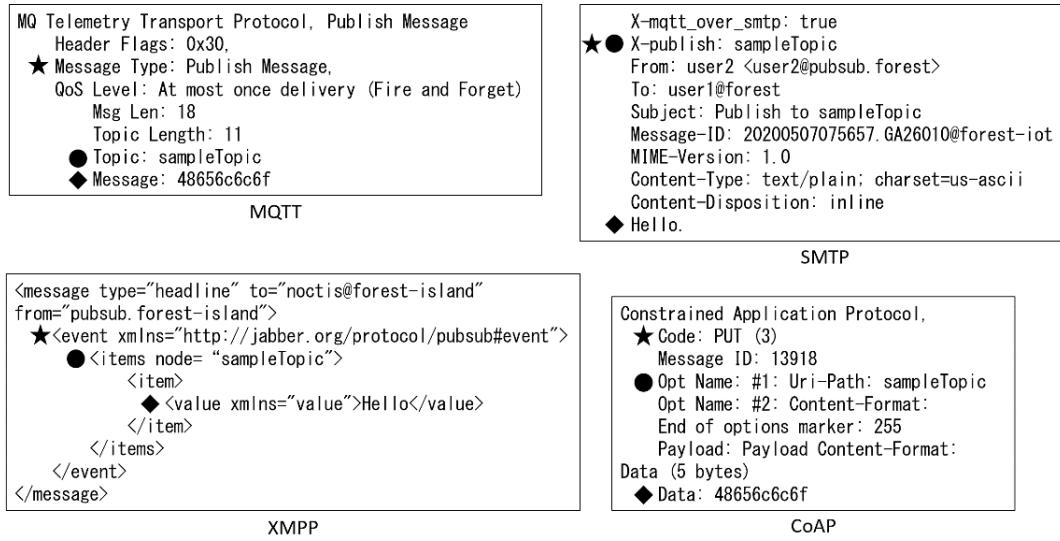


図 20. 各フィールドの対応関係

### 4.3 異なるユーザ設定におけるリクエスト処理位置の変更

ディストリビュータにピギーバック機能を実装し、プローカ機能の負荷分散を実験した。図 21 に示すように IoT デバイスとクラウドの中間に3層のディストリビュータ(A, B, C)が配置されている環境を想定し、5 台を連続に有線接続した。IoT デバイスはクラウドと TCP 接続を確立してバーストリクエストを送信した。通信の始めはすべてのリクエストがディストリビュータ A の担当であると仮定し局所的な負荷を発生させた。ディストリビュータ A, B がピギーバックした負荷情報はディストリビュータ C に伝えられた。ディストリビュータ C では、伝達された負荷情報をもとにユーザが定義した式によって各ディストリビュータにおけるアプリケーションの負荷を知ることができる。ディストリビュータ C はダウンストリームに制御情報をピギーバックすることでディストリビュータ A, B のリクエスト処理量を調整し負荷を分散させた。その際、ユーザのアプリケーション設計によって各ディストリビュータにどの程度処理を割り振るべきか異なることを考慮した。3台のディストリビュータにはほぼ均等にリクエスト処理を実行させる場合と、あるディストリビュータにできるだけ多くリクエスト処理を実行させる場合を想定し、柔軟にリクエスト処理位置が変更できるかどうかを実験した。また、ピギーバックによる占有帯域の増加量と処理遅延、負荷の応答性を測定した。プローカの要求遅延は数 100ms 程度と考え、ディストリビュータがピギーバックする頻度を 100ms に 1 回とした。バーストリクエストの詳細を表 2 に示す。およそ 50 万 packet per second (PPS), 1Gbps, 10000TCP コネクションだった。



図 21. ピギーバックによる負荷分散実験のデバイス構成

表 2. バーストリクエストの詳細

平均PPS	486,494 PPS
Bandwidth	927.91 Mbps
TCP Connections	10,247

#### 4.4 リクエスト処理位置の変更による負荷分散の性能比較

ピギーバックを用いた負荷分散と、ピギーバックを用いない負荷分散を比較した。実験環境は4.3と同様に図21に示すような環境であり、3層のディストリビュータを想定した。IoTデバイスは表2に従うリクエストを発生させた。ピギーバックを用いない手法ではIoTデバイスから見てディストリビュータによるインフラはネットワーク透過な環境ではないため、IoTデバイスはディストリビュータとコネクションを確立しリクエストを送信した。IoTデバイスのバーストリクエストによって高負荷になったディストリビュータAはそれ以上のリクエストを拒否し、IoTデバイスに対しディストリビュータB,Cを利用するように通達した。通達を受信したIoTデバイスは自らディストリビュータB,Cに再接続しリクエストを送信した。ピギーバックを用いた手法とピギーバックを用いない手法の負荷分散にかかる時間を測定した。

#### 4.5 通信の一貫性を保ったマイグレーションによる負荷分散

リクエスト処理位置の変更に加え、ピギーバックを用いてディストリビュータ間のアプリケーションのマイグレーション機能を実装した。上位から下位のディストリビュータへのマイグレーション機能は3.2で示した手順に従って実装した。実験環境はリクエスト処理位置の変更実験と同様に図21に示すネットワーク構成だった。上位のディストリビュータ(B)から下位のディストリビュータ(A)にプローカをマイグレーションし、IoTデバイスから見て通信の一貫性を維持できるか確かめた。

## 5 結果

### 5.1 アプリケーションプロトコルの変換

CoAP による PUT を、MQTT のパブリッシュ、XMPP のパブリッシュ、X-Header を用いた SMTP のパブリッシュに変換した。図 22 から図 25 にそれぞれのプロトコルのパケットを Wireshark[30] で表示した様子を示す。CoAP では”sampleTopic”という URI に”Hello”という Data を PUT したことが分かる。ディストリビュータによって作成された MQTT のパブリッシュ（図 23）では “sampleTopic”と“Hello”はそれぞれ Topic と Message に、XMPP のパブリッシュ（図 24）ではそれぞれ node と ITEM 内の value に割り当てられた。図 25 に示す SMTP では X-mqtt\_over\_smtp と X-publish によってパブリッシュである旨が挿入されており、Subject（メール件名）には汎用的な文章が書き込まれたことが分かる。ディストリビュータが IoT デバイス側に送信したパケットは、MQTT のサブスクライブや CoAP の PUT の対象であるクラウドを送信元として作成された。Wireshark 画面上部にあるキャプチャしたパケットの一覧を見ると TCP 通信の sequence 番号や acknowledge 番号の不一致による再送信ではなく、TCP 通信のシーケンスに矛盾がなかったことがわかる。このため中間にディストリビュータが存在しないように見え、IoT デバイスには一切手を加えることなく異なるプロトコルでデータ連携をすることができた。また、学術情報ネットワーク（SINET）と西研究室の共同研究においてコアネットワークのある地点でインターネットを観測したところ、1 時間に約 200 万個の TCP ストリームが存在することが分かった。ディストリビュータは IoT デバイス-クラウド間の TCP 通信やアプリケーションプロトコルのステートの管理をメモリ上で行い、そのサイズはおよそ 3-4KB だった。したがって仮にすべてのトラフィックがプロトコル変換を行ったとしても必要なメモリ領域は約 6-8GB 程度であり、汎用のベアボーン型 PC で搭載可能なメモリ使用量であることが分かる。

No.	Time	Source	Destination	Protocol	Length	Info
8	0.002715	192.168.200.30	192.168.200.5	MQTT	84	Subscribe Request (id=1) [sampleTopic]
9	0.003701	192.168.200.5	192.168.200.30	MQTT	71	Subscribe Ack (id=1)
10	0.003708	192.168.200.30	192.168.200.5	TCP	66	52141 → 1883 [ACK] Seq=33 Ack=10 Win=1
11	5.330968	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:61055, PUT, /sampleTopic
12	5.331432	192.168.200.5	192.168.200.30	MQTT	74	Publish Message [sampleTopic]
13	5.331458	192.168.200.30	192.168.200.5	TCP	66	52141 → 1883 [ACK] Seq=33 Ack=30 Win=1
14	7.362432	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:61055, PUT, /sampleTopic
15	7.362890	192.168.200.5	192.168.200.30	MQTT	74	Publish Message [sampleTopic]
16	7.362918	192.168.200.30	192.168.200.5	TCP	66	52141 → 1883 [ACK] Seq=33 Ack=50 Win=1
17	11.424176	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:61055, PUT, /sampleTopic
18	11.424622	192.168.200.5	192.168.200.30	MQTT	74	Publish Message [sampleTopic]
▶ Frame 14: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)						
▶ Ethernet II, Src: Elitegro_67:f5:9e (f4:4d:30:67:f5:9e), Dst: EliteGro_a4:0e:3a (94:c6:91:a4:0e:3a)						
▶ Internet Protocol Version 4, Src: 192.168.200.30, Dst: 192.168.200.5						
▶ User Datagram Protocol, Src Port: 43037, Dst Port: 5683						
▼ Constrained Application Protocol, Confirmable, PUT, MID:61055						
01.. .... = Version: 1						
..00 .... = Type: Confirmable (0)						
.... 0000 = Token Length: 0						
Code: PUT (3)						
Message ID: 61055						
> Opt Name: #1 Uri-Path: sampleTopic						
> Opt Name: #2: Content-Format: application/octet-stream						
End of options marker: 255						
[Uri-Path: /sampleTopic]						
> Payload: Payload Content-Format: application/octet-stream, Length: 5						
▼ Data (5 bytes)						
Data: 48656cc6c6f						
[Length: 5]						
0000 94 c6 91 a4 0e 3a f4 4d 30 67 f5 9e 08 00 45 00 . ....:M 0g....E.						
0010 00 34 18 58 40 00 40 11 10 ec c0 a8 c8 1e c0 a8 .4 X@. ....						
0020 c8 05 a8 1d 16 33 00 20 11 a7 40 03 ee 7f bb 73 ..3 ..@....						
0030 61 6d 70 6c 65 54 6f 70 69 63 11 2a ff 48 65 6c ampleTopic ic.*.Hello						
0040 6c 6f 1o						
put2mqtt.pcap						

図 22. CoAP による PUT の詳細

No.	Time	Source	Destination	Protocol	Length	Info
8	0.002715	192.168.200.30	192.168.200.5	MQTT	84	Subscribe Request (id=1) [sampleTopic]
9	0.003701	192.168.200.5	192.168.200.30	MQTT	71	Subscribe Ack (id=1)
10	0.003708	192.168.200.30	192.168.200.5	TCP	66	52141 → 1883 [ACK] Seq=33 Ack=10
11	5.330968	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:61055, PUT, /sampleTopic
12	5.331432	192.168.200.5	192.168.200.30	MQTT	74	Publish Message [sampleTopic]
13	5.331458	192.168.200.30	192.168.200.5	TCP	66	52141 → 1883 [ACK] Seq=33 Ack=30
14	7.362432	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:61055, PUT, /sampleTopic
15	7.362898	192.168.200.5	192.168.200.30	MQTT	74	Publish Message [sampleTopic]
16	7.362918	192.168.200.30	192.168.200.5	TCP	66	52141 → 1883 [ACK] Seq=33 Ack=50
17	11.424176	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:61055, PUT, /sampleTopic

> Frame 15: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)  
> Ethernet II, Src: Shuttle\_e0:0c:30 (80:ee:73:e0:0c:30), Dst: Elitegno\_67:f5:9e (f4:4d:30:67:f5:9e)  
> Internet Protocol Version 4, Src: 192.168.200.5, Dst: 192.168.200.30  
> Transmission Control Protocol, Src Port: 1883, Dst Port: 52141, Seq: 30, Ack: 33, Len: 20  
▼ MQ Telemetry Transport Protocol, Publish Message  
  ▼ Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)  
    0011 .... = Message Type: Publish Message (3)  
     .... 0... = DUP Flag: Not set  
     .... .0. = QoS Level: At most once delivery (Fire and Forget) (0)  
     .... ..0 = Retain: Not set  
    Msg Len: 18  
    Topic Length: 11  
    Topic: sampleTopic  
    Message: 48656c6c6f  
0000 f4 4d 30 67 f5 9e 80 ee 73 e0 0c 30 08 00 45 00 ·M0g... s..0..E.  
0010 00 3c 40 00 00 40 06 29 47 c0 a8 c8 05 c0 a8 <@...@:G.....  
0020 c8 1e 07 5b cb ad a1 c6 2d b1 c9 4e ad 50 10 ...[....-N P.  
0030 05 78 da c2 00 00 30 12 00 0b 73 61 6d 70 6c 65 x...0..sample  
0040 54 6f 70 69 63 48 65 6c 6c 6f TopicHel lo

put2mqtt.pcap

図 23. 変換された MQTT パブリッシュ

No.	Time	Source	Destination	Protocol	Length	Info
91	9.044276	192.168.200.30	192.168.200.5	XMP/x...	171	IQ(get) QUERY(jabber:iq:roster)
92	9.045617	192.168.200.5	192.168.200.30	XMP/x...	297	IQ(result) QUERY(jabber:iq:roster)
93	9.045624	192.168.200.30	192.168.200.5	TCP	66	36446 → 5222 [ACK] Seq=894 Ack=1717
94	9.049122	192.168.200.30	192.168.200.5	XMP/x...	92	PRESENCE
95	9.050508	192.168.200.5	192.168.200.30	XMP/x...	156	PRESENCE < noctis@forest-island/b27
96	9.050518	192.168.200.30	192.168.200.5	TCP	66	36446 → 5222 [ACK] Seq=920 Ack=1807
97	11.828467	192.168.200.30	192.168.200.5	CoAP	66	CON, MID:39882, PUT, /sampleTopic
98	11.828938	192.168.200.5	192.168.200.30	XMP/x...	287	MESSAGE < pubsub森林-island
99	11.828961	192.168.200.30	192.168.200.5	TCP	66	36446 → 5222 [ACK] Seq=920 Ack=2040

> Frame 98: 287 bytes on wire (2296 bits), 287 bytes captured (2296 bits)  
> Ethernet II, Src: Shuttle\_e0:0c:30 (80:ee:73:e0:0c:30), Dst: Elitegno\_67:f5:9e (f4:4d:30:67:f5:9e)  
> Internet Protocol Version 4, Src: 192.168.200.5, Dst: 192.168.200.30  
> Transmission Control Protocol, Src Port: 5222, Dst Port: 36446, Seq: 1807, Ack: 920, Len: 233  
▼ XMPP Protocol  
  ▼ MESSAGE [type="headline"]  
    from: pubsub.forest-island  
    to: noctis@forest-island  
    type: headline  
  ▼ EVENT [xmlns="http://jabber.org/protocol/pubsub#event"] [UNKNOWN]  
    xmlns: http://jabber.org/protocol/pubsub#event  
  ▼ ITEMS [node="sampleTopic"]  
    ▼ node: sampleTopic [UNKNOWN ATTR]  
      > [Expert Info (Note/Undecoded): Unknown attribute node]  
    ▼ ITEM  
      ▼ VALUE [xmlns="value"]  
        xmlns: value  
        CDATA: Hello  
      > [Expert Info (Note/Undecoded): Unknown element: event]  
0000 f4 4d 30 67 f5 9e 80 ee 73 e0 0c 30 00 00 45 00 ·M0g... s..0..E.  
0010 01 11 48 00 00 00 40 06 28 72 c0 a8 c8 05 c0 a8 ..@...@:(r.....  
0020 c8 1e 44 66 5e 5e a0 ef 97 20 bd 28 be 50 10 ...f.^... .(P.  
0030 05 78 7b ea 00 00 3c 6d 65 73 73 61 67 65 20 74 x{...<m message t  
0040 79 70 65 3d 22 68 65 61 64 6c 69 6e 65 22 20 74 ype="hea dline" t  
0050 6f 3d 22 6e 6f 63 74 69 73 49 66 6f 72 65 73 74 o="nocti s@forest  
0060 2d 69 73 6c 61 6e 64 22 20 66 72 6f 6d 3d 22 70 -island" from="p  
0070 75 62 73 75 62 2e 66 6f 72 65 73 74 2d 69 73 6c ubsub.fo rest-isl  
0080 61 6e 64 22 3e 3c 65 76 65 74 20 78 6d 6c 6e and"><ev ent xmlns  
0090 73 3d 22 68 74 74 70 3a 2f 6a 61 62 62 65 72 s="http://jabber  
00a0 2e 6f 72 67 2f 70 72 6f 74 6f 63 6f 2f 70 75 .org/protocol/pu  
00b0 62 73 75 62 23 65 76 65 66 74 22 3e 3c 69 74 65 bsub#event"><ite  
00c0 6d 73 20 6e 6f 64 65 3d 22 73 61 6d 70 6c 65 54 ms node="sampleT  
00d0 6f 70 69 63 22 3e 3c 69 74 65 6d 3e 3c 76 61 6c opic"><i tem=<val  
00e0 75 65 20 78 6d 6c 6e 73 3d 22 76 61 6c 75 65 22 ue xmlns ="value"  
00f0 3e 48 65 6c 6c 6f 3c 2f 76 61 6c 75 65 3e 3c 2f >Hello</ value></  
0100 69 74 65 6d 3e 3c 2f 69 74 65 6d 73 3e 3c 2f 65 item>< i tems></e  
0110 76 65 6e 74 3e 3c 2f 6d 65 73 73 61 67 65 3e vent></m message>

put2xmpp.pcap

図 24. 変換された XMPP パブリッシュ

```

No. Time Source Destination Protocol Length Info
22 0.037875 192.168.1.20 192.168.1.30 SMTP 105 C: DATA fragment, 39 bytes
23 0.037882 192.168.1.30 192.168.1.20 TCP 66 25 → 46114 [ACK] Seq=121 Ack=134 Wi
24 0.038165 192.168.1.20 192.168.1.30 SMTP/I... 374 from: root <morishima@forest-island>
25 0.038171 192.168.1.30 192.168.1.20 TCP 66 25 → 46114 [ACK] Seq=121 Ack=442 Wi
26 0.042709 192.168.1.30 192.168.1.20 SMTP 103 S: 250 2.0.0 Ok: queued as E79212C0

> Frame 24: 374 bytes on wire (2992 bits), 374 bytes captured (2992 bits)
> Ethernet II, Src: Buffalo_9:a:b2:26 (34:3d:c4:9a:b2:26), Dst: Buffalo_2:e:17:4c (18:c2:bf:2e:17:4c)
> Internet Protocol Version 4, Src: 192.168.1.20, Dst: 192.168.1.30
> Transmission Control Protocol, Src Port: 46114, Dst Port: 25, Seq: 134, Ack: 121, Len: 308
> Simple Mail Transfer Protocol
  < Internet Message Format
    Date: Sun, 27 Dec 2020 21:39:59 +0900
    < From: root <morishima@forest-island>, 1 item
      > Item: root <morishima@forest-island>\r\n
    < To: morishima@forest-iot, 1 item
      > Item: morishima@forest-iot\r\n
    Subject: Publish message for sampleTopic
    Message-ID: <20201227123959.GA6150@forest-door0>
    MIME-Version: 1.0
    Content-Type: text/plain; charset=us-ascii
    Unknown-Extension: Content-Disposition: inline (Contact Wireshark developers if you want this supported.)
    Unknown-Extension: X-publish: sampleTopic (Contact Wireshark developers if you want this supported.)
      Type: X-publish
      Value: sampleTopic
    Unknown-Extension: X-mqtt_over_smtp: true (Contact Wireshark developers if you want this supported.)
      Type: X-mqtt_over_smtp
      Value: true
    Line-based text data: text/plain (1 lines)
      Hello\r\n

0000  44 61 74 65 3a 20 53 75 6e 2c 20 32 37 20 44 65 Date: Su, 27 De
0010  63 20 32 30 32 30 20 32 31 3a 33 39 3a 35 39 20 c 2020 2 1:39:59
0020  2b 39 30 30 0d 0a 46 72 6f 6d 3a 20 72 6f 6f +0900 -F rom: roo
0030  74 20 3c 6d 7f 69 73 68 69 6d 61 40 66 6f 72 t <morishima@for
0040  65 73 74 2d 69 73 6c 61 6e 64 3e 0d 0a 54 6f 3a est-isla nd> -To:
0050  20 6d 6f 72 69 73 68 69 6d 61 40 66 6f 72 65 73 morishima@fores
0060  74 2d 69 6f 74 0d 0a 53 75 62 6a 65 63 74 3a 20 t-iot -S ubject:
0070  50 75 62 6c 69 73 68 20 6d 65 73 73 61 67 65 20 Publish message
0080  66 6f 72 20 73 61 6d 70 6c 65 54 6f 70 69 63 0d for samp leTopic
0090  0a 4d 65 73 73 61 67 65 2d 49 44 3a 20 3c 32 30 .Message-ID: <20
00a0  32 30 31 32 32 37 31 32 33 39 35 39 2e 47 41 36 20122712 3959.GA6
00b0  31 35 39 48 66 6f 72 65 73 74 2d 64 6f 72 30 150@fore st-door0
00c0  3e 0d 0a 4d 49 4d 45 2d 56 65 72 73 69 6f 6e 3a > -MIME-Version:
00d0  20 31 2e 30 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 1.0 -Co ntent-Ty
00e0  70 65 3a 20 74 65 78 74 2f 70 6c 61 69 6e 3b 20 pe: text /plain;
00f0  63 68 61 72 73 65 74 3d 75 73 2d 61 73 63 69 69 charset= us-ascii
0100  0d 0a 43 6f 6e 74 65 6e 74 2d 44 69 73 70 6f 73 ..Conten t-Dispos
0110  69 74 69 6f 6e 3a 20 69 6e 6c 69 6e 65 0d 0a 58 ition: i nline -X
0120  2d 70 75 62 6c 69 73 68 3a 20 73 61 6d 70 6c 65 -publish : sample
0130  54 6f 70 69 63 0d 0a 58 2d 6d 71 74 74 5f 6f 76 Topic: X -mqtt_over_
0140  65 72 5f 73 6d 74 70 3a 20 74 72 75 65 0d 0a 0d er_smtp: true...
0150  0a 48 65 6c 6f 0d 0a 0d Hello... -Hello...

```

Frame (374 bytes) Reassembled SMTP (344 bytes)

put2smtp.pcap

図 25. X-Header を用いて変換された SMTP

## 5.2 異なるユーザ設定におけるリクエスト処理位置の変更

実験で用いたプローラの負荷を表現する式は次に示すとおりである。Linux が提供するロードアベレージ、CPU 使用率、メモリ使用量を考慮しつつ、プローラに対する単位時間当たりのリクエスト数(*pps*)に重きを置いた。

$$Load = 0.4 \times pps + 0.2 \times load\_average + 0.2 \times cpu\_usage + 0.2 \times mem\_usage$$

図 26 に 3 台のディストリビュータの負荷を平滑化する設定にした場合の結果を示す。時間が 10 秒の時点でのバーストリクエストを発生させた。始め、リクエストはすべてディストリビュータ A が処理をしたためディストリビュータ A の負荷が上昇した。ディストリビュータ C は自身の負荷情報と、ピギーバックによって伝達されたディストリビュータ A, B の負荷情報から各ディストリビュータのリクエスト処理量を変更した。3 台のディストリビュータの負荷がおおよそ均等に分散されたことが分かった。図 27 は 3 台のディストリビュータの内、ディストリビュータ A にできるだけ多くのリクエストを処理させた場合の結果である。ディストリビュータ A が全負荷の約 6 割を処理し、残りの

リクエストをディストリビュータ B, C で分担した。これらの結果から、ユーザの設定によって各ディストリビュータの負荷を均等に分散させるだけでなく、あるディストリビュータに余力を持たせておき突発的な負荷上昇に備えるような負荷分散ができることが分かった。また、図 26 および図 27 を見ると時刻 9 秒～11 秒における負荷の急激な変化を表現することができていることから、プロ一ヵのようなアプリケーションでは 100ms のピギーバックは十分な粒度であることが分かった。

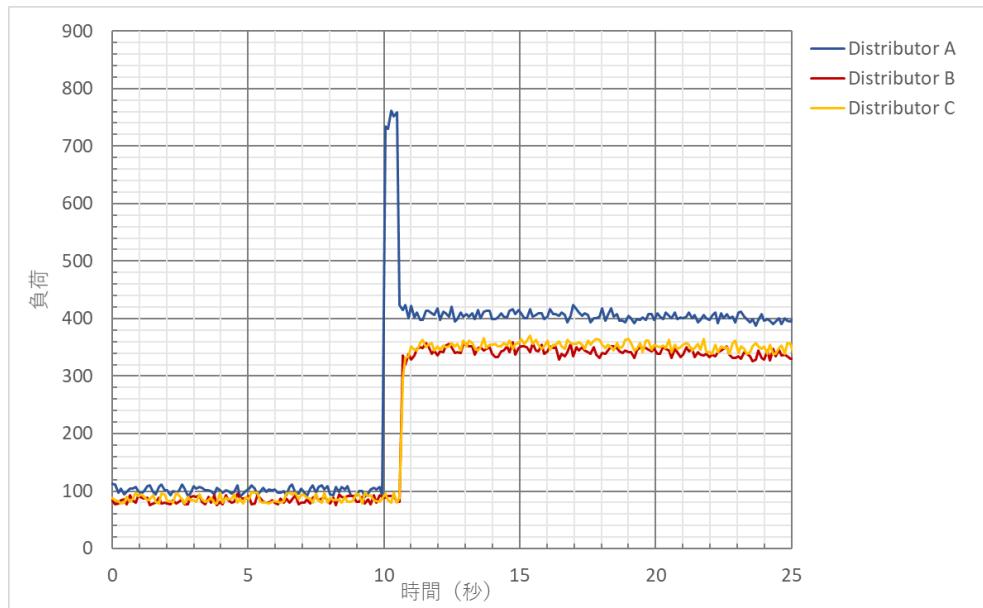


図 26. 3 台のディストリビュータの負荷を平滑化する場合の負荷の推移

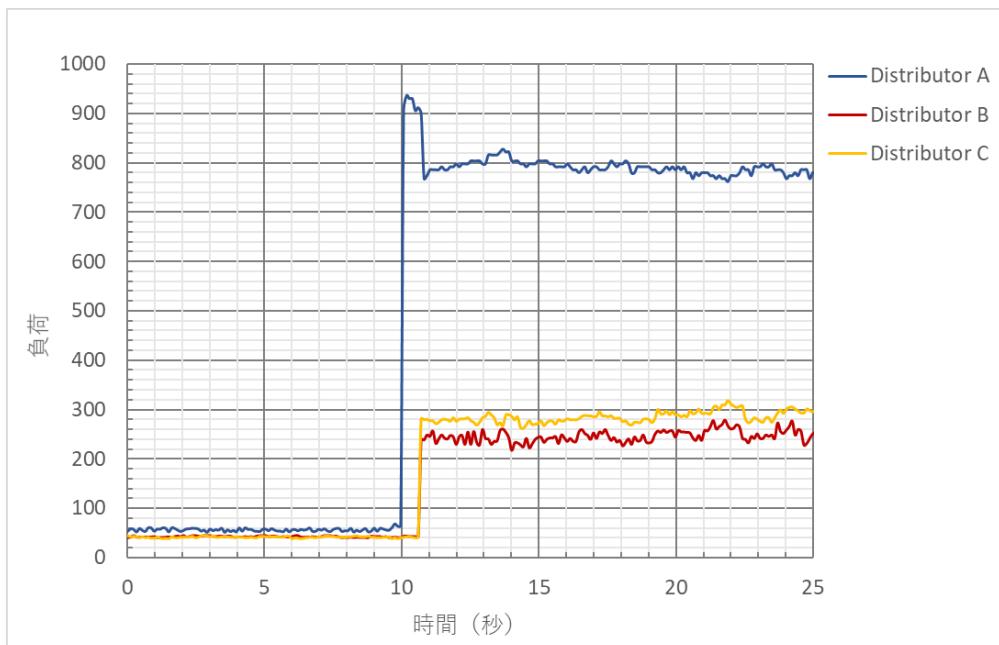


図 27. ディストリビュータ A ができるだけ多くのリクエストを処理した場合の負荷の推移

図 28 は図 26 において負荷分散が実行された約 1 秒間を切り出した図である。負荷が大きく変化した時点の時刻をグラフ下側に表記した。ディストリビュータがピギーバックする頻度が 100ms に 1 回だったためグラフにおける負荷の打点もおよそ 100ms ごとに行われた。時刻 10.471 秒から 10.580 秒の間にディストリビュータ C からのピギーバックによってディストリビュータ A のリクエスト処理量が変更されたことが分かる。その後、時刻 10.580 秒から 10.684 秒の間にディストリビュータ A に処理されていたリクエストの一部がフォワードされディストリビュータ B, C の負荷の上昇が起きた。最終的に負荷がディストリビュータ B, C に分散されたことが見えたのは時刻 10.684 秒の時点だった。

また、ピギーバックのサイズを 20byte としたときのディストリビュータがピギーバックする頻度と占有帯域の関係を図 29 に示す。ピギーバックをしない場合の占有帯域は 926.83Mbps であり、ピギーバックを 100ms に 1 回行った場合は 926.996Mbps (0.175% 上昇)、10ms に 1 回行った場合は 926.998Mbps (0.177% 上昇) だった。このことから、ピギーバックが占有帯域に与える影響は非常に小さいことが分かった。表 3 はピギーバックの付与や削除にかかる処理遅延をまとめた表である。ピギーバックの付与は  $14.25\mu s$ 、パースと削除はそれぞれ  $2.09\mu s$ ,  $1.07\mu s$  の処理遅延だった。流れるパケットにピギーバックを付与する作業は新しいメモリ領域を確保する必要があるためピギーバックのパースと削除より長い処理時間を要した。ネットワーク遅延を含めたピギーバックによる処理遅延の影響を計算する。例えば実験環境のように通信経路上にディストリビュータが 3 台存在すると仮定する。下位 2 つのディストリビュータがピギーバックを付与し、最上位のディストリビュータがピギーバックのパース及び削除を行ったとすると、ピギーバックの処理遅延は

$$14.25\mu s + 14.25\mu s + 2.09\mu s + 1.07\mu s = 31.66\mu s$$

となる。IoT デバイス-クラウド間の往復遅延を 50ms とすると往復遅延の増加は

$$\frac{50ms + 31.66\mu s}{50ms} = \frac{50,000\mu s + 31.66\mu s}{50,000\mu s} = 1.0006332$$

であるから、増加率は約 0.06% である。ピギーバックの処理遅延はネットワーク遅延と比較して僅かであることが分かった。

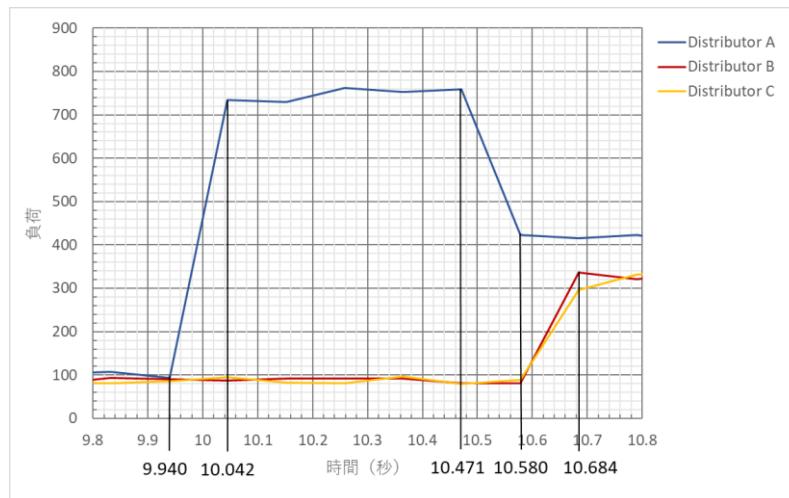


図 28. 負荷分散時の拡大図

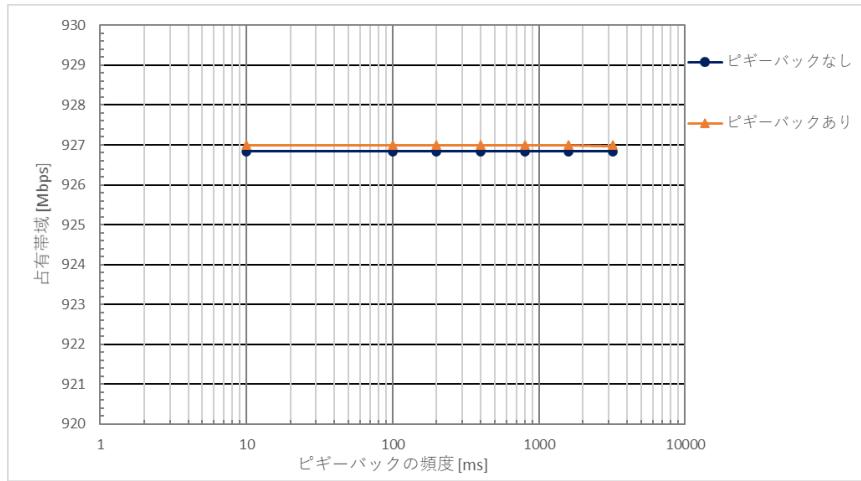


図 29. ピギーバックする頻度と占有帯域の関係

表 3. ピギーバックに関する処理の遅延

処理内容	要した時間
ピギーバックの付与 (append)	$14.25 \mu s$
ピギーバックのパース	$2.09 \mu s$
ピギーバックの削除 (trim)	$1.07 \mu s$

### 5.3 リクエスト処理位置の変更による負荷分散の性能比較

5.2 で示したピギーバックを用いたリクエスト処理位置の変更による負荷分散の結果を再掲する。また、図 30 にピギーバックを用いない負荷分散の結果を示す。どちらも時刻 10 秒の時点でのバーストリクエストが発生した。表 4 は両者の負荷分散に関する性能を比較した表である。ディストリビュータ A にバーストリクエストが集中していた時間はピギーバックを用いた手法で約 0.53 秒、ピギーバックを用いない手法で約 1.38 秒だった。パケットに負荷の情報をピギーバックし上流にフォワードする操作と比べて、ピギーバックを用いない手法ではディストリビュータ A が IoT デバイスに指示を送信する必要があったため負荷が集中していた時間が長くなった。また、バーストリクエストが発生してから実際にディストリビュータ B, C にリクエストが分散されるまでの時間はピギーバックを用いた場合は約 0.64 秒、ピギーバックを用いない場合は約 3.73 秒だった。ピギーバックを用いることによって IoT デバイスは TCP 通信を切断することなくリクエストを送信することができるが、ピギーバックを用いない手法では IoT デバイスはディストリビュータ B, C に TCP 通信を再接続した。プロトコルスタックの実行と TCP 通信のハンドシェイクによるオーバーヘッドによってリクエスト送信先が変更されるまでに約 3.73 秒を要した。また、ピギーバックを用いない手法では IoT デバイスが接続するディストリビュータを変更するのに要した約 2.35 秒、IoT デバイスはセンサデータをパブリッシュすることができなかった。

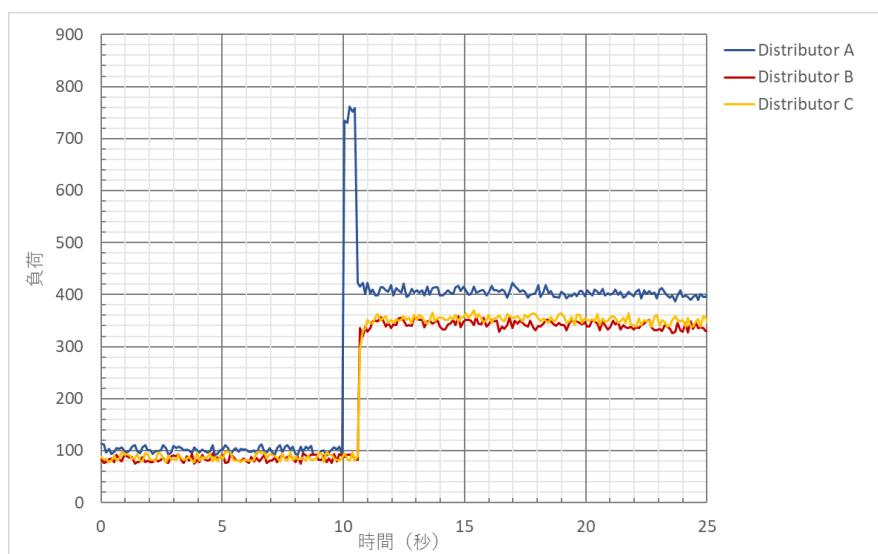


図 26. 3 台のディストリビュータの負荷を平滑化する場合の負荷の推移

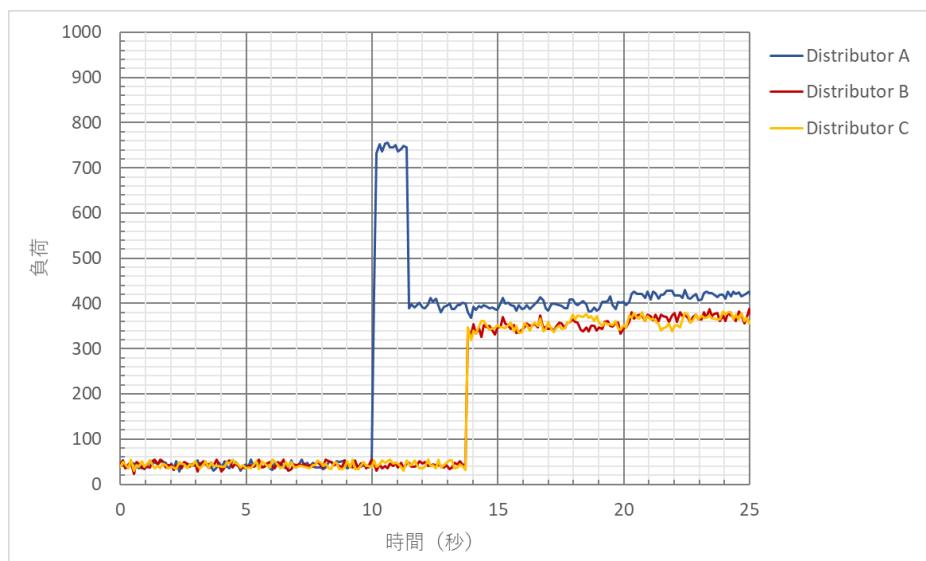


図 30. ピギーバックを用いない負荷分散による負荷の推移

表 4. 負荷分散に関する性能比較

	ピギーバック有	ピギーバック無
Aにリクエストが集中していた時間	0.53 sec	1.38 sec
B,Cへの負荷分散にかかった時間	0.64 sec	3.73 sec

## 5.4 通信の一貫性を保ったマイグレーションによる負荷分散

図 31 にディストリビュータ B からディストリビュータ A にブローカをマイグレーションした際の負荷の変化を示す。時刻 20.45 秒にマイグレーションが開始し、IoT デバイスからのリクエストはディストリビュータ A にバッファリングされた。マイグレーションが完了しバッファリングされたリクエストがアプリケーションに処理され始めたのは時刻 21.94 秒の時点だった。IoT デバイス-クラウド間通信の一貫性を維持したか確認するために、ディストリビュータ A, B においてリクエストを処理した際に TCP 通信の sequence 番号を表示した。図 32 の上半分はディストリビュータ B が処理したパケットの sequence 番号を、下半分はディストリビュータ A が処理したパケットの sequence 番号を、上から下に順に並べて示している。ディストリビュータ B がマイグレーションを実行する前に処理した最後のパケットの sequence 番号は 105987992 番であり、ペイロードサイズは 100byte だった。マイグレーション完了後、ディストリビュータ A が最初に処理したパケットは 105988092 番だった。

$$105987992 + 100 = 105988092$$

であるから、ディストリビュータ A はマイグレーションを実行する前に処理されたパケットに連続するパケットを処理できたことが分かる。IoT デバイスから見るとマイグレーション中に TCP 接続は切断することなく、通信を続行することができた。

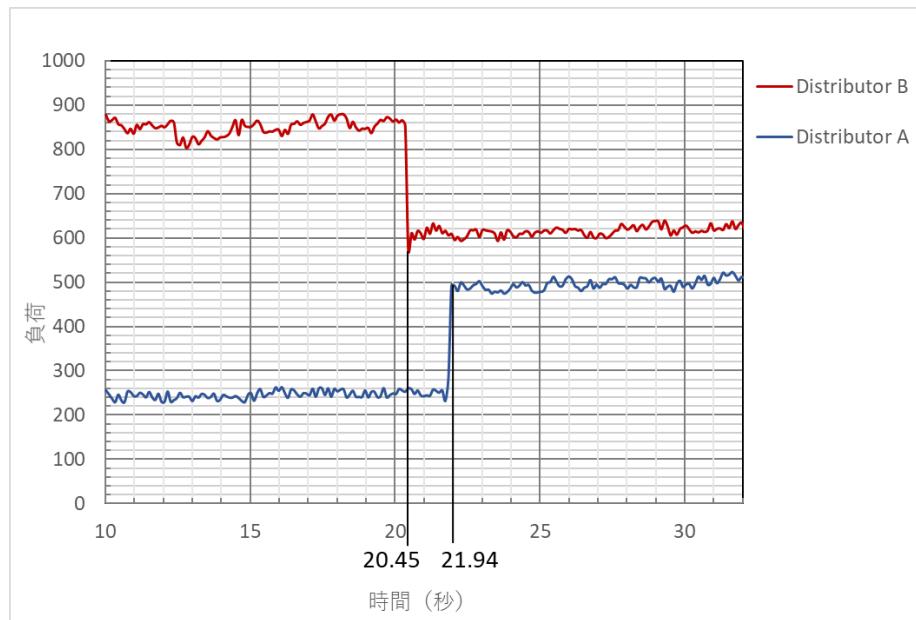


図 31. マイグレーションによる負荷の推移

```
seq number: 105986592 payload size: 100  
seq number: 105986692 payload size: 100  
seq number: 105986792 payload size: 100  
seq number: 105986892 payload size: 100  
seq number: 105986992 payload size: 100  
seq number: 105987092 payload size: 100  
seq number: 105987192 payload size: 100  
seq number: 105987292 payload size: 100  
seq number: 105987392 payload size: 100  
seq number: 105987492 payload size: 100  
seq number: 105987692 payload size: 100  
seq number: 105987792 payload size: 100  
seq number: 105987892 payload size: 100  
seq number: 105987992 payload size: 100  
Flag migration done  
+-----+  
seq number: 105988092 payload size: 200  
seq number: 105988292 payload size: 1400  
seq number: 105989692 payload size: 100  
seq number: 105989792 payload size: 100  
seq number: 105989892 payload size: 100  
seq number: 105989992 payload size: 100  
seq number: 105990092 payload size: 100  
seq number: 105990192 payload size: 100  
seq number: 105990292 payload size: 100  
seq number: 105990392 payload size: 100  
seq number: 105990492 payload size: 100  
seq number: 105990592 payload size: 100  
seq number: 105990692 payload size: 100  
seq number: 105990792 payload size: 100  
seq number: 105990892 payload size: 100  
+-----+
```

図 32. 各ディストリビュータが処理したパケットの sequence 番号

## 6 結論と将来の展望

本研究ではスマートコミュニティにおいてエッジ、フォグ領域でネットワーク透過性を持つディストリビュータと呼ぶノード上に多様なアプリケーションが展開される環境を想定した。ディストリビュータ間の効率的な情報伝達手段としてエンドデバイス-クラウド間の通信にディストリビュータの負荷情報や他のディストリビュータに対する制御情報をピギーバックすることを提唱した。ピギーバックによって交換された負荷情報はアプリケーション作成者に提供され、リクエスト処理位置の変更やアプリケーションのマイグレーションを既存のインフラを変更することなく実行する。また、ディストリビュータがネットワーク途中でプロトコル変換を行うことで IoT の相互運用性を確保する。IoT デバイスおよびエンドユーザはパブリッシュ/サブスクライブ形式や REST 形式など、様々なプロトコルを用いてアプリケーションを利用することができる。

ピギーバックの機能やアプリケーションプロトコルの変換を評価するために、西研究室が開発を進めている DooR と汎用のベアボーン型 PC を用いてプロトコル機能を実装し、実験を行った。アプリケーションプロトコルの変換実験では、IoT デバイスに一切変更を加えることなくディストリビュータ上でプロトコルを変換することができた。ピギーバックを用いた実験では、ディストリビュータがユーザ設定によってアプリケーションの負荷を表現しリクエスト処理位置を柔軟に変更できること、ピギーバックが占有帯域と処理遅延に与える影響が非常に小さかったことを確認した。また、ディストリビュータが有するネットワーク透過性によって、IoT デバイスは負荷分散時でも通信を再接続することなくセンサデータを送信することができた。ピギーバックの応用実験では、パケットにマーキングすることで通信の一貫性を維持したアプリケーションのマイグレーションを達成した。

本研究ではある1つの通信経路上のディストリビュータ間のみ考慮し負荷を分散したが、複数の通信経路を考慮した負荷分散も考えることができる。エッジ、フォグ領域の上位層に位置するネットワークの分岐点にあるディストリビュータは自身に接続する複数経路上の下位ディストリビュータの負荷をまとめて監視することができる。この上位層のディストリビュータは、複数の通信経路にとって共通の計算資源であるため、自身の仕事量を調節することによって複数経路間で負荷を分散させることができる。例えば通信経路 A, B の分岐点に位置するディストリビュータは通信経路 A の負荷が増加した場合、通信経路 B の仕事を一時的に下位ディストリビュータに担当させることで、通信経路 A の仕事を請け負うことができる。また、さらに視野を広げ、ディストリビュータによるネットワーク全体の負荷とアプリケーションの要求を考慮した負荷分散の最適化も今後の本研究の発展内容である。

## 7 参考文献

- [1] H. NISHI, “Infrastructure and Services for Smart Community,” *The Journal of the Institute of Electronics, Information, and Communication Engineers*, vol. 98, no. 2, pp. 112-117, Feb. 2015.
- [2] N. Naik and P. Jenkins, “Web protocols and challenges of Web latency in the Web of Things,” in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2016, pp. 845-850, doi: 10.1109/ICUFN.2016.7537156.
- [3] N. Naik, “Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP,” in *2017 IEEE International Systems Engineering Symposium (ISSE)*, Oct. 2017, pp. 1-7, doi: 10.1109/SysEng.2017.8088251.
- [4] “Home | AMQP.” <https://www.amqp.org/> (accessed Dec. 05, 2020).
- [5] “MQTT – The Standard for IoT Messaging.” <https://mqtt.org/> (accessed Dec. 05, 2020).
- [6] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP).” <https://tools.ietf.org/html/rfc7252> (accessed Dec. 05, 2020).
- [7] P. Saint-Andre <[psaintan@cisco.com](mailto:psaintan@cisco.com)>, “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.” <https://tools.ietf.org/html/rfc6121> (accessed Dec. 05, 2020).
- [8] E. C. P. Neto, G. Callou, and F. Aires, “An algorithm to optimise the load distribution of fog environments,” in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2017, pp. 1292-1297, doi: 10.1109/SMC.2017.8122791.
- [9] A. Yousefpour, G. Ishigaki, and J. P. Jue, “Fog Computing: Towards Minimizing Delay in the Internet of Things,” in *2017 IEEE International Conference on Edge Computing (EDGE)*, Jun. 2017, pp. 17-24, doi: 10.1109/IEEE.EDGE.2017.12.
- [10] R. Beraldì, C. Canali, R. Lancellotti, and G. P. Mattia, “Distributed load balancing for heterogeneous fog computing infrastructures in smart cities,” *Pervasive and Mobile Computing*, vol. 67, p. 101221, Sep. 2020, doi: 10.1016/j.pmcj.2020.101221.
- [11] R. Beraldì, H. Alnuweiri, and A. Mtibaa, “A Power-of-Two Choices Based Algorithm for Fog Computing,” *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 698-709, Jul. 2020, doi: 10.1109/TCC.2018.2828809.
- [12] R. Beraldì and H. Alnuweiri, “Exploiting Power-of-Choices for Load Balancing in Fog Computing,” in *2019 IEEE International Conference on Fog Computing (ICFC)*, Jun. 2019, pp. 80-86, doi: 10.1109/ICFC.2019.00019.
- [13] “A Cooperative Fog Approach for Effective Workload Balancing – IEEE Journals & Magazine.” <https://ieeexplore.ieee.org/document/7912240> (accessed Nov. 30, 2020).
- [14] “[PDF] Bridging MQTT & XMPP Internet of Things networks | Semantic Scholar.” <https://www.semanticscholar.org/paper/Bridging-MQTT-%26-XMPP-Internet-of-Things-networks-Waher-Laboratorios/31fbe3a5eaafdf5f8671213e906cc005f01217e5e> (accessed Dec. 03, 2020).
- [15] M. Dave, J. Doshi, and H. Arolkar, “MQTT– CoAP Interconnector: IoT Interoperability Solution for Application Layer Protocols,” in *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Oct. 2020, pp. 122-127, doi: 10.1109/I-SMAC49090.2020.9243377.
- [16] G. Bouloukakis, N. Georgantas, P. Ntumba, and V. Issarny, “Automated synthesis of

- mediators for middleware-layer protocol interoperability in the IoT,” *Future Generation Computer Systems*, vol. 101, pp. 1271–1294, Dec. 2019, doi: 10.1016/j.future.2019.05.064.
- [17] “IoT Interoperability—On-Demand and Low Latency Transparent Multiprotocol Translator – IEEE Journals & Magazine.” <https://ieeexplore.ieee.org/abstract/document/7908961> (accessed Dec. 04, 2020).
- [18] K. Saito and H. Nishi, “Application Protocol Conversion Corresponding to Various IoT Protocols,” in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2020, pp. 5219–5225, doi: 10.1109/IECON43393.2020.9255101.
- [19] W. A. S. P. Abeysiriwardhana, J. Wijekoon, R. L. Tennekoon, and H. Nishi, “Software-accelerated Service-oriented Router for Edge and Fog Service Enhancement Using Advanced Stream Content Analysis,” *IEEJ Transactions on Electronics, Information and Systems*, vol. 139, no. 8, pp. 891–899, 2019, doi: 10.1541/ieeejeiss.139.891.
- [20] T. Miura, J. L. Wijekoon, S. Prageeth, and H. Nishi, “Novel infrastructure with common API using docker for scaling the degree of platforms for smart community services,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, Jul. 2017, pp. 474–479, doi: 10.1109/INDIN.2017.8104818.
- [21] “Industrial Edge Computing: Enabling Embedded Intelligence – IEEE Journals & Magazine.” <https://ieeexplore.ieee.org/abstract/document/8941000> (accessed Nov. 22, 2020).
- [22] “Home,” *DPDK*. <https://www.dpdk.org/> (accessed Dec. 22, 2020).
- [23] R. L. Tennekoon, “Implementation and evaluation of secured network infrastructure using content-based router,” PhD Thesis, Keio University, 2019.
- [24] “BurntSushi/aho-corasick,” *GitHub*. <https://github.com/BurntSushi/aho-corasick> (accessed Dec. 25, 2020).
- [25] INTEL CORPORATION, “Hyperscan,” *01.org*, Sep. 17, 2015. <https://01.org/hyperscan> (accessed Jul. 07, 2017).
- [26] “Delivering 160Gbps DPI Performance on the Intel Xeon Processor E5-2600 Series using HyperScan,” p. 6.
- [27] “Shuttle Global – DH310.” <http://global.shuttle.com/products/productsDetail?productId=2261> (accessed Dec. 06, 2020).
- [28] “Intel® NUC Kit NUC6i3SYK Product Specifications.” <https://ark.intel.com/content/www/us/en/ark/products/89186/intel-nuc-kit-nuc6i3syk.html> (accessed Dec. 26, 2020).
- [29] “Intel® NUC Kit NUC8i5BEK Product Specifications.” <https://ark.intel.com/content/www/us/en/ark/products/126147/intel-nuc-kit-nuc8i5bek.html> (accessed Dec. 25, 2020).
- [30] “Wireshark · Go Deep.” <https://www.wireshark.org/> (accessed Jan. 05, 2021).

## 謝辞

研究を進めるにあたり、3年間ご指導、ご鞭撻を賜りました西宏章教授に心より御礼申し上げます。また、西裕子様は日々の研究室生活や学会発表など多くの場面で学生を支えてくださいました。

研究活動にあたり、発表等で研究に関するご意見、ご指摘を頂きました中山直明教授、寺岡文男教授、大槻知明教授、金子晋丈准教授にこの場で深く御礼申し上げます。本論文を書くにあたり、査読をしてくださった齋藤賢太君、野原将人君にも感謝を申し上げます。また、三年間研究室生活を送るなかでお世話になった先輩、同期、後輩の方々にも改めて深く感謝を申し上げます。新型コロナウイルスが収束し、また一堂に会することを心から願ってやみません。

最後に、謝辞まで読んでくださった読者の皆様に感謝致します。

2021年1月7日