

P4 を用いた MQTT-SN アクセラレーション

大澤 昂生[†] 日比 智也[‡] 坂野 遼平[†]

[†] 工学院大学 情報学部 情報通信工学科 〒192-0015 東京都八王子市中野町 2665-1

[‡] NTT 未来ねっと研究所 〒239-0847 神奈川県横須賀市光の丘 1-1

E-mail: [†] {j017048@ns, banno@cc}.kogakuin.ac.jp, [‡] tomoya.hibi.ha@hco.ntt.co.jp

あらまし IoT の主要な通信プロトコルとして MQTT が存在する。MQTT では Broker と呼ばれるサーバを介してすべての通信が行われる。そのため、Broker への負荷集中が懸念されている。本研究では UDP 上で動作する MQTT-SN を対象として、P4 を用いて Broker の処理をネットワーク内にオフロードする手法を提案する。P4 はデータプレーンをプログラム可能な言語であり、提案手法では一定の条件を満たした Publish メッセージをネットワークスイッチから直接 Subscriber に配信することで Broker の負荷を軽減する。P4 のリファレンス実装である BMv2 に提案手法を実装し、評価実験を行った。評価結果から、Broker に到達するパケット数を低減できることが明らかになった。また、遅延時間に関しては 100 Byte の Publish メッセージ送信時に 40% 以上の遅延時間削減が確認できた。

キーワード P4, データプレーンプログラマビリティ, MQTT, MQTT-SN, IoT

MQTT-SN Acceleration using P4

Koki OSAWA[†] Tomoya HIBI[‡] and Ryohei BANNO[†]

[†] Department of Information and Communications Engineering, Faculty of Informatics, Kogakuin University
2665-1 Nakano-machi, Hachioji-shi, Tokyo, 192-0015 Japan

[‡] NTT Network Innovation Laboratories, 1-1 Hikarinooka, Yokosuka-shi, Kanagawa, 239-0847 Japan

E-mail: [†] {j017048@ns, banno@cc}.kogakuin.ac.jp, [‡] tomoya.hibi.ha@hco.ntt.co.jp

Abstract MQTT is a major communication protocol for the IoT. In MQTT, all communication is done through a server called a "broker". Therefore, there is a concern that the broker is overloaded. In this study, we focus on MQTT-SN which is based on UDP and propose a method to offload the processing of the broker into the network using P4. P4 is a language for data plane programming, and the proposed method reduces the load on the broker by delivering publish messages that satisfy certain conditions directly from the network switch to the corresponding subscribers. We implemented the proposed method in BMv2, a reference implementation of P4, and conducted evaluation experiments. The experimental results show that the proposed method can reduce the number of packets that reach the broker. In addition, we confirmed that the proposed method reduces the delay time by more than 40% when sending a 100 Byte publish message.

Keywords P4, Data Plane Programmability, MQTT, MQTT-SN, IoT

1. はじめに

IoT (Internet of Things) デバイスが世界的に急速に普及している。総務省の情報通信統計では 2018 年に全世界で 307.1 億台の IoT デバイスが稼働していると統計づけられ、2021 年には 400 億台を超えると見込まれている[1]。IoT デバイスの通信プロトコルに MQTT (MQ Telemetry Transport) [2] が存在する。MQTT は Publish/Subscribe 型の通信プロトコルである。この方式ではセンサデータ等のデータを提供するクライアント (Publisher) が Broker と呼ばれるサーバにデータを発行 (Publish) し、Broker は対象データを購読 (Subscribe) しているクライアント (Subscriber) にデータの転送を行う。よって、IoT システムの規模が大

きくなればなるほど Broker への負荷集中が懸念される[3]。

本研究では、MQTT の派生プロトコルである MQTT-SN [4] を対象とし、Broker の一部機能を Broker 直前のネットワーク内にて行うことで、Broker の使用リソースの削減と周辺ネットワークでのトラフィックの削減を行うことを目的とする。

2. 既存技術

2.1. MQTT

2.1.1. 概要

MQTT とは IoT システムで多く用いられる Publish/Subscribe 型の通信プロトコルである。ヘッダサイズが軽量であり、IoT や M2M (Machine to Machine)

といったメモリやネットワーク帯域が限られる環境での利用に適している。この通信アーキテクチャは図 1 のように Publisher, Broker 及び Subscriber の 3 種類のエンティティから成り立っている。

Publisher はセンサデータなどを Publish するクライアントであり、Topic を指定してセンサデータなどの Publish を行う。Subscriber はデータを受信するクライアントであり、あらかじめ Topic を指定して Subscribe を行う。Broker は Publisher からの Publish を管理し、該当する Topic を Subscribe している Subscriber に配信を行う。

2.1.2. QoS

MQTT において QoS (Quality of Service) は 0, 1, 2 と 3 種類の設定ができる。QoS に関してのまとめたものを表 1 に示す。QoS 0 ではクライアント・サーバ間の通信は最高 1 回送信される。この設定では Publish メッセージに対する確認応答が無いため、メッセージの到達は保証されない。次に QoS 1 では、クライアント・サーバ間の通信は最低 1 回送信が行われる。Publish メッセージに対し確認応答として PubAck を返すためメッセージが相手に届くことが保障されるが、重複してメッセージが送信されてしまう可能性がある。最後に QoS 2 では、クライアント・サーバ間の通信が正確に 1 回のみ送信される。Publish メッセージに対し PubRec, PubRel, PubComp と相互に 2 回通信を行うため、重複したメッセージは届かない。

2.1.3. Retain

Retain とはトピック毎の最新メッセージを保持する機能である。Publish メッセージには Retain 用のフラグが存在し、このフラグの立っている Publish メッセージは Broker 上で保持される。Broker が新たに Subscribe メッセージを受信した際、保持されたメッセージが存在する場合 Subscriber に Publish される。Retain フラグがない場合、次に Publisher からデータが送られてこない限りデータは新規 Subscriber に送信されることはない。Retain の動作順序を説明したものを図 2 に示す。

2.2. MQTT-SN (MQTT-Sensor Network)

MQTT-SN とは UDP 上で動作する通信プロトコルであり、基本動作は MQTT と変わらない。MQTT は本来 TCP 上で動作する通信プロトコルである。しかし、IoT システムは様々な条件のもと稼働しており TCP の動作が不向きな場面も存在する。例えば劣悪なネットワーク環境である場合や電源に制約のある環境である場合などが挙げられる。TCP を使用すると再送制御が多数発生し、更にネットワーク状況を悪くしかねない。TCP を使用すると再送制御が多数発生し、更にネットワーク状況を悪くしかねない。

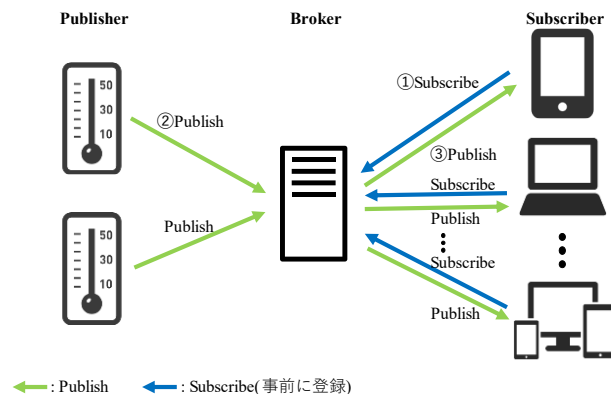


図 1 MQTT 概要図

表 1 MQTT の QoS

QoS	メッセージおよび方向	意味
0	Publish →	- 最高1回 (At most once) 送信 - メッセージが送信先に確実に届くのかの保証はされない
1	Publish → ← PubAck	- 最低1回 (At least once) 送信 - メッセージが送信先に届くと保証 - 重複してメッセージが届く可能性
2	Publish → ← PubRec → PubRel ← PubComp	- 正確に1回 (Exactly once) 送信 - メッセージが送信先に届くと保証 - 重複したメッセージが届く可能性はない

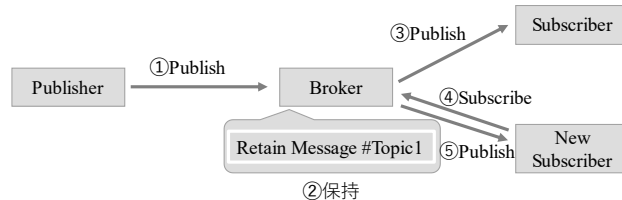


図 2 Retain 動作フロー

また、TCP ではネットワークへの常時接続が基本となるが、それでは制限されている電源を無駄に使ってしまうこととなる。そのような場合、UDP ベースのプロトコルは優位といえる。

2.3. データプレーンプログラマビリティ

2.3.1. 概要

データプレーンプログラマビリティでは通信機器の動作をプログラム可能で、その動作をアプリケーションから詳細に制御可能である。現在様々な研究が行われ、これらの取り組みに DPN (Deeply Programmable Networks) [5]や P4 (Programming Protocol-Independent Packet Processors) 言語[6]が存在する。性能面の特徴としてはアプリケーションの一部を通信機器内に実装することで低遅延化などの恩恵を受けられると期待されている。

2.3.2. P4

P4 はデータプレーンをプログラムするための言語である。SDN (Software Defined Networking) にてよく用いられる OpenFlow [7]ではスイッチのコントロールを行うコントロールプレーンに関しては制御可能であった。しかし、実際にデータ通信を行うデータプレー

ンは既存のプロトコルの設定をはめ込んで使う形式である。これは新しいプロトコルや、自定義のプロトコルに OpenFlow が使えない要因である。一方 P4 では、コントロールプレーンだけでなく、データプレーンまでも自身で記述し制御することが可能である。このようなことから、大手のデータセンタ・通信事業者・通信機器ベンダ・ASIC ベンダなどに注目を浴び開発が進められている。

3. 提案手法

本研究では、MQTT-SN を対象として Broker の一部処理をネットワーク内にオフロードする手法を提案する。ネットワーク内に P4 言語対応のスイッチを置き、そのスイッチに Broker の一部機能を実装する。この Broker の一部動作を実装した P4 スwitch をアクセラレータと呼称する。

提案手法の概要を図 3 に示す。アクセラレータでは、3 つの動作パターンを想定している。最初に、Subscribe メッセージを受け取った場合である。この場合、アクセラレータは P4 コントローラと連携して IP アドレスや Topic 名などの情報の記録・保持を行う。次に指定条件に合致する Publish メッセージを受け取った場合である。この指定条件とは、QoS レベルが 0 であり Retain フラグの立っていない Publish メッセージである。この場合、Subscribe メッセージ受信時に記録したデータベースを基に Broker を介さずに Subscriber に対し Publish メッセージの配信を行う。最後にこれらの条件以外のものに関してである。この場合は、アクセラレータにて制御を加えてしまうと Broker の動作に悪影響を与える場合もあると考えられる為、従来通りルーティングを行う。

アクセラレータの実装に関しては、Subscribe 部はスタティックな設定を用いて、それ以外の部分は P4 にて実装した。本研究において作成した P4 プログラムの動作をまとめたフローチャートを図 4 に示す。

4. 評価

4.1. 評価方法

評価には図 5 に示すトポロジを用いた。P4 スwitch におけるアクセラレータ機能の有無を切り替えながら、各ホストの送受信パケット数および遅延時間を測定した。Publisher が送信する Publish メッセージについては、QoS レベルを 0, Retain フラグを無効と設定した。なお、Publisher が Publish メッセージを送信し始める前に Subscriber は Subscribe メッセージを Broker に宛て送信している。

評価は Windows の PC 上にインストールされた VirtualBox の上で動作する Ubuntu16.04.7 にて行った。VM は P4 言語公式 GitHub [8]にて提供されているものを利用した。

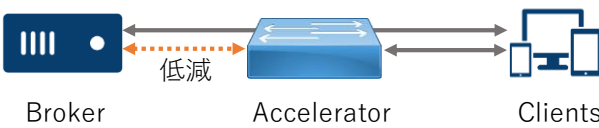


図 3 提案手法概要

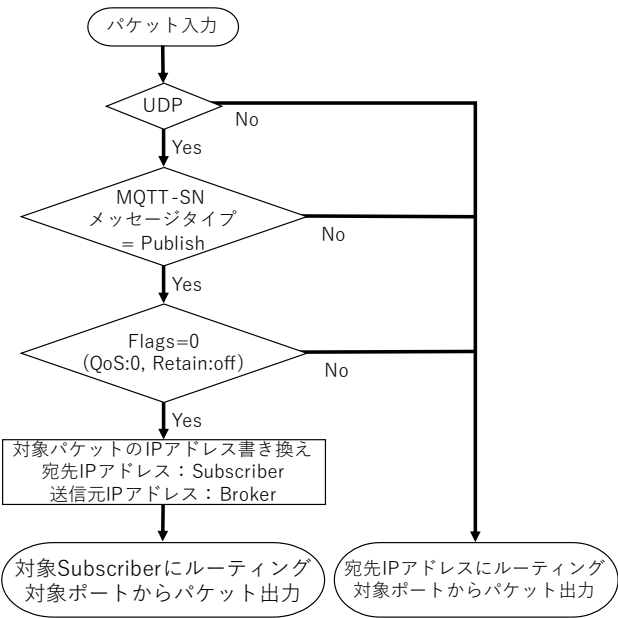


図 4 アクセラレータ動作フロー

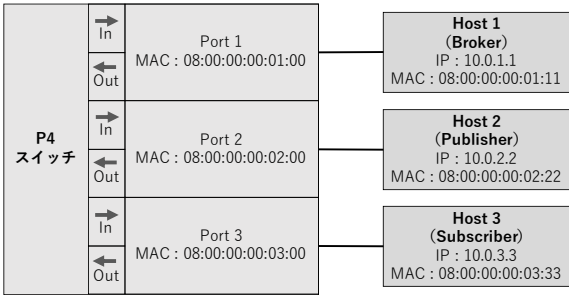


図 5 実験トポロジ

表 2 Windows PC 環境

項目	内容
CPU	Intel Core i5-10400 6core/12thread @2.90GHz
RAM	32GB
OS	Windows 10 Pro Education 64bit
VM Software	Oracle VirtualBox Version 6.1.12 r139181 (Qt5.6.2)

表 3 VM 環境

項目	内容
割り当てCPU	2 Core
割り当てRAM	2GB
OS	Ubuntu 16.04.7 LTS
Network Analyzer	Wireshark 2.6.10
Broker Software	Mosquitto RSMB (Really Small Message Broker) Version 1.3.0.2
Client Software	mqtt-sn-tools Vresion 0.0.6
P4 Software Switch	BMv2 Version 1.13.0
Network Emulator	Mininet Version 2.3.0b6

この VM では P4 のソフトウェアスイッチである BMv2 が使用可能である． 3 章で述べた提案手法を BMv2 に実装し，同 VM 上の Mininet にて図 5 のトポロジにて評価を行った．Broker には RSMB (Really Small Message Broker) [9]を利用し，クライアントには mqtt-sn-tools [10]を改変して用いた．使用した環境について，表 2 に VM を動かしている Windows PC の環境を，表 3 に VM 周りの環境設定および使用ソフトウェアを，それぞれ示す．

具体的な評価方法を解説する．まずは送受信パケット数の測定についてである．P4 スイッチのそれぞれのポートを出入りするパケットは Wireshark にて記録を行っている．送受信パケットの測定に関してはこの Wireshark のデータを基に算出した．送受信パケット数に関しては Subscriber が Broker に対し Subscribe を行うパケットも評価対象としている．つまり，Subscriber が Subscribe 動作を行うところから Publisher が接続を切るまでを評価対象とした．Publisher の Publish メッセージ数は $10 \cdot 100 \cdot 1000 \cdot 10000 \cdot 100000$ とし，これらは連続した評価でなく，例えば Publish メッセージ数 10 の評価が終了したら，スイッチやホストを再起動してから次の評価を行った．これらの Publish メッセージは 1 ミリ秒ごとに Publisher から送信される．

次に遅延測定に関して解説する．遅延実験では Publish メッセージ数を 100000 回と固定した．Publisher が Publish メッセージを作成した時刻をマイクロ秒単位で取得し，このタイムスタンプを Publish メッセージのペイロードとしている．なお，合計のパケットサイズが 100 Byte もしくは 200 Byte になるようにペイロードをパディングしている．Subscriber 側では Publish メッセージ受信時の時刻をマイクロ秒単位で取得し，Publish メッセージに含まれる送信時刻情報と合わせて記録した．Subscriber 作成のタイムスタンプから Publisher 作成のタイムスタンプを減算することで遅延時間をマイクロ秒単位で算出した．評価に用いたパラメータを表 4 に示す．

4.2. 評価結果

まずは，送受信パケットに関しての評価の結果を示す．表 5 に Publish メッセージ数が 100 の時の従来手法および提案手法の送受信パケット数を示す．また，表 5 をもとに従来手法および提案方式の評価結果をグラフ化したものをそれぞれ図 6，図 7 に示す．従来手法と提案手法を比較すると Publisher および Subscriber の送受信パケット数は変化がないことが見て取れる．しかし，Broker の送受信パケット数は提案手法が 100 パケット分少なくなっている．図 8，9，10 は，Publish メッセージ数を 10 から 100000 まで変化させた際の送受信パケット数を示している．

表 4 評価パラメータ

	送受信パケット数測定	遅延測定
Publish数	10/100/1000/10000/100000	100000
QoS (Publish)	0	0
QoS (Subscribe)	0	0
Retain	Off	Off
パケットサイズ	100 Byte	100 Byte/200 Byte

表 5 送受信パケット数測定結果 (100 Publish)

	P4スイッチ					
	Broker		Publisher		Subscriber	
	in	out	in	out	in	out
従来手法	106	106	3	103	103	3
提案手法	6	6	3	103	103	3

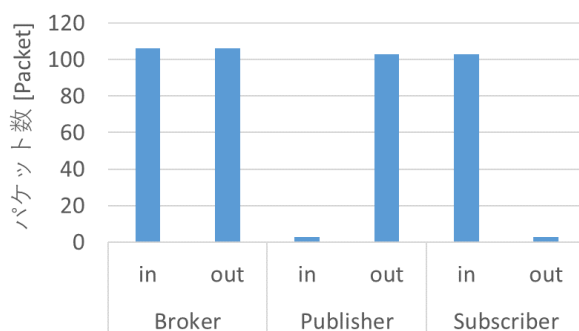


図 6 ホストごとのパケットの送受信(従来手法)

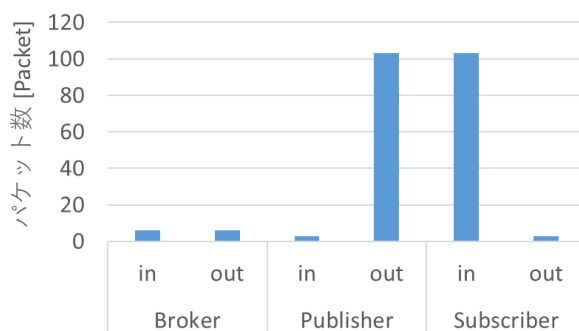


図 7 ホストごとのパケットの送受信(提案手法)

Publish メッセージ数が 100 の時と同じく，Publisher および Subscriber の送受信パケット数は従来手法と提案手法ともに大差がないことが見て取れる．図 8 の Broker のグラフからは従来手法に関しては，送信側受信側の線が重なり，Publish メッセージ数が増えるとともに線形的に通過パケット数も上昇している．一方，提案手法に関しては Publish メッセージ数が 10 から 100000 と増加しても通過パケット数は約 10 付近で落ち着いていることが分かる．

次に遅延測定に関しての評価である．図 11 に遅延測定結果のグラフを示す．本評価では，100000 パケットの遅延の平均および中央値を比較した．BMv2 は P4 のリファレンス実装であるが，ハードウェア製品と比較すると性能が不安定であり外れ値が発生しやすい．

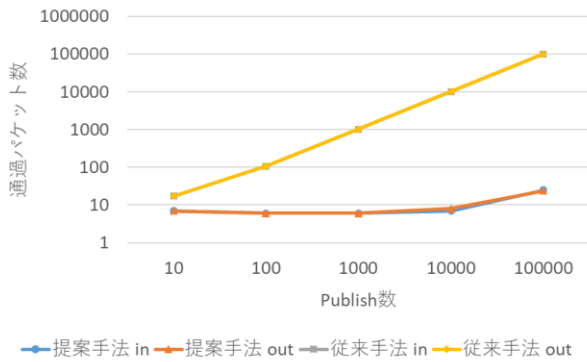


図 8 Publish 数に対する送受信パケット数の変化 (Broker)

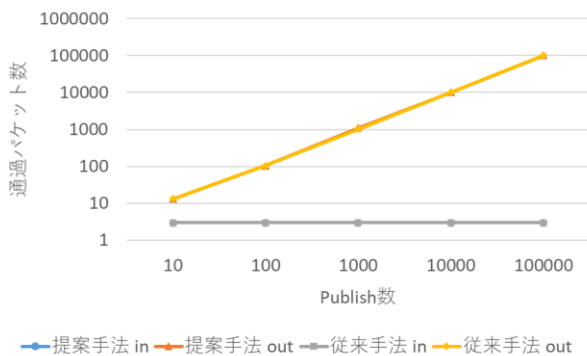


図 9 Publish 数に対する送受信パケット数の変化 (Publisher)

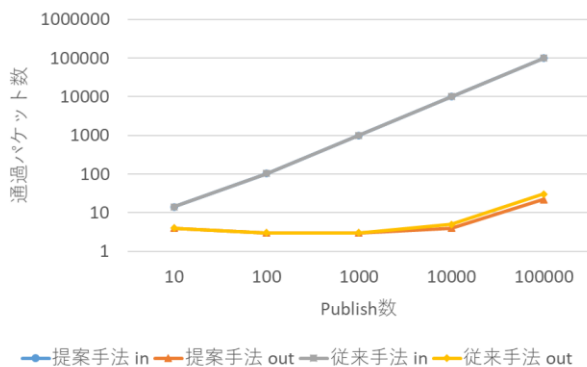


図 10 Publish 数に対する送受信パケット数の変化 (Subscriber)

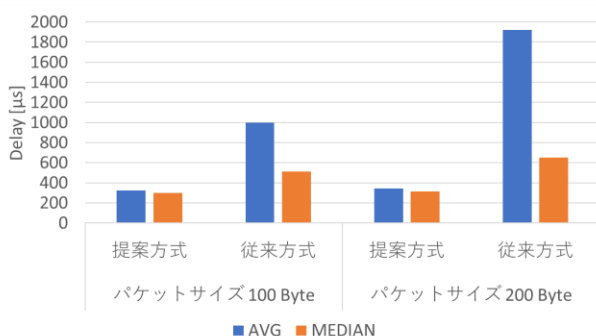


図 11 遅延測定結果

このため、平均値に加え、外れ値の影響を受け難い中央値も算出した。100 Byte および 200 Byte どちらの場合においても提案方式は中央値、平均値両方において短い遅延時間で伝送できていることがわかる。また、平均値と中央値の差に着目すると、提案方式では従来方式と比べ差が少ないことが見て取れる。

5. 考察

送受信パケット数の評価結果より、Publisher および Subscriber の送受信パケット数を比較するとほぼ一致していることが分かった。Broker の送受信パケット数を比較すると提案手法と従来手法のパケット数に大きな開きがある。表 5 から提案手法の送受信パケット数がちょうど 100 少ないことが分かる。同様に図 7 から Publish メッセージ数と比例して提案手法は送受信パケット数が少ないことが分かる。実際に Wireshark のデータを見てみたところ提案手法では Publish メッセージが Broker に届いておらず、アクセラレータによって Subscriber に配信されていることが確認できた。このことからアクセラレータは適切に Broker の処理を代行できていると言える。Broker が受信する Publish メッセージが削減されることで、Broker の処理負荷を軽減でき、また周辺ネットワークの利用量も低減できる。

次に、遅延測定の考察を行う。パケットサイズが 100 Byte の時、従来方式は中央値で約 512 マイクロ秒、提案方式は中央値で約 300 マイクロ秒の遅延であった。さらに、パケットサイズが 200 Byte の場合、従来方式は中央値で約 653 マイクロ秒、提案方式は中央値で約 314 マイクロ秒の遅延であった。このことから、それぞれ 41.4%、51.9%の遅延時間の短縮が図れている。アクセラレータが適切に稼働していることにより、アクセラレータから Broker まで、および Broker からアクセラレータまでの伝送時間が削減されたことが、この遅延短縮の大きな要因であると考えられる。BMv2 の GitHub [11]にて、製品グレードではなく開発・検証向けのツールであると述べられている。今回の評価から平均値と中央値の差が大きく開いたのも BMv2 に起因するものだと考えられる。さらに従来方式の方が平均値と中央値の差が開いているということに関しては、BMv2 に流れるパケット数が提案方式と比べ Publish メッセージの分、多いことから増加しているのではないかと考えられる。

6. おわりに

本研究では、IoT システムに使われる通信プロトコル MQTT-SN において Broker の機器リソースの削減、および周辺ネットワークでのトラフィック削減を目的とした。この目的に対し本研究では P4 を用いて Broker の処理の一部をネットワーク内で行う手法を提案した。P4 ソフトウェアスイッチ BMv2 に Broker の一部処理

を行わせるよう実装し、Mininet 上で評価を行った。評価から、Broker の送受信パケットの削減が確認でき、また遅延時間の短縮にも効果があることを確認した。以上より、研究目的である Broker の使用リソースの削減、周辺ネットワークにおけるトラフィックの削減に関して、本提案方式は有効な手段であることを確認した。

謝辞 本研究の一部は、JSPS 科研費 19K20253 の支援を受けて行われたものである。

文 献

- [1] 総務省, "令和元年版情報通信白書", July 2019.
- [2] OASIS, "MQTT Version 3.1.1 ", <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>, 閲覧日 2020/12/23
- [3] Wiriyang Pipatsakulroj, Vasaka Visoottiviseth, Ryousei Takano, "muMQ: A Lightweight and Scalable MQTT Broker", IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pp. 1-6, July 2017.
- [4] mqtt.org, "MQTT-SNv1.2 ", https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf, 閲覧日 2020/12/23
- [5] 蕪澤 慎之介, 原 雅貴, 中尾 彰宏, 小口 正人, 山本 周, 山口 実靖, "スイッチにおけるペイロード解析に基づくアプリケーションの高速化", 信学技報, vol. 115, no. 483, NS2015-177, pp. 53-57, 2016 年 3 月.
- [6] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghes, David Walker, "P4: programming protocol-independent packet processors", ACM SIGCOMM Computer Communication Review, Volume 44, Issue 3, pp.5-131, July 2014.
- [7] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner, "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review, Volume 38, Issue 2, pp.65-68, March 2008.
- [8] github.com/p4lang, "P4 Tutrial", <https://github.com/p4lang/tutorials>, 閲覧日 2020/12/23
- [9] github.com/eclipse, "mosquito rsmb" <https://github.com/eclipse/mosquitto.rsmb>, 閲覧日 2020/12/23
- [10] github.com/njh, "mqtt-sn-tools" <https://github.com/njh/mqtt-sn-tools>, 閲覧日 2020/12/23
- [11] github.com/p4lang, "behavioral-model" <https://github.com/p4lang/behavioral-model>, 閲覧日 2020/12/23