一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
IA2024-15(2024-09)

分散MQTTブローカーにおける Bloom filterを用いたワイルドカードトピック管理

坂野　遼平†　　渡辺　良人††,†

† 一橋大学 大学院ソーシャル・データサイエンス研究科　〒186–8601 東京都国立市中 2–1
†† Solid Surface 株式会社　〒101–0054 東京都千代田区神田錦町 1–13
E-mail: †banno@computer.org, ††yoshito.watanabe@solidsurface.co.jp

あらまし　MQTT は Publish/Subscribe 型の通信プロトコルであり，特に IoT システムにおいて広く用いられている．MQTT による通信ではブローカーにメッセージ処理負荷が集中することから，大規模システムにおいては複数ブローカーを用いた負荷分散が必要となる．その際，ブローカー間でトピック情報を共有することで，連携のオーバヘッドを低減し，スループットを向上可能であることが知られている．共有するトピック情報は膨大な量となる可能性があるため，Bloom filter を用いてコンパクトに保持し，トピックの有無を検索可能とする手法が提案されている．しかしながら，既存手法では，トピックに多様なワイルドカードが含まれる場合に，Bloom filter に対するクエリ数が増大する問題がある．こうしたクエリ数の増大は，ブローカーのメッセージ処理性能に悪影響を及ぼす可能性がある．本研究では，Bloom filter によるトピック情報管理において，ワイルドカードを効率的に扱う手法を提案する．提案手法ではトピックの接頭部を Bloom filter に登録し，検索時には存在し得るワイルドカードのパターンを考慮しつつ Trie と類似した方式でクエリを処理する．シミュレーション実験により，特に PUBLISH メッセージにマッチするトピックが存在しない場合等において，提案手法が既存手法と比べ Bloom filter に対するクエリ数を大幅に低減可能であることが明らかとなった．

キーワード　MQTT, Publish/subscribe, Bloom filter, IoT

Managing Wildcard Topics by Bloom Filter in Distributed MQTT Brokers

Ryohei BANNO† and Yoshito WATANABE††,†

† Graduate School of Social Data Science, Hitotsubashi University　2–1, Naka, Kunitachi, Tokyo, 186–8601 Japan
†† Solid Surface Inc.　1-13, Kanda-Nishikicho, Chiyoda-ku, Tokyo, 101–0054 Japan
E-mail: †banno@computer.org, ††yoshito.watanabe@solidsurface.co.jp

Abstract　MQTT is a widely utilized protocol in IoT systems. Due to the concentration of messages on an MQTT broker, large-scale systems require load distribution across multiple brokers. For efficient load distribution, a common approach to prevent message flooding is to share subscription topics among them. Since the size of topic information could be large, some existing methods use a Bloom filter to store and manage the topics in a space-efficient manner. However, these techniques have a problem in handling wildcards, as searching for a single topic may require numerous queries to the Bloom filter. In this paper, we present a method for efficiently managing wildcard topics using a Bloom filter. Our proposed method involves adding prefixes of subscription topics to the Bloom filter. Searching for a published topic resembles using a Trie structure, while it considers wildcard patterns. Experimental results show that, in many cases, our method reduces the number of queries to the Bloom filter compared to existing approaches.

Key words　MQTT, Publish-subscribe, Bloom filter, IoT

1. はじめに

IoT システムにおいて広く用いられる通信プロトコルとして，MQTT [1] がある．Publish/Subscribe 型の通信モデル [2] に基づいており，クライアント間でやり取りされるメッセージをブローカーが仲介することで，高い疎結合性を得られる利点がある．MQTT においてメッセージを送信するクライアントはパブリッシャー，受信するクライアントはサブスクライバーと呼

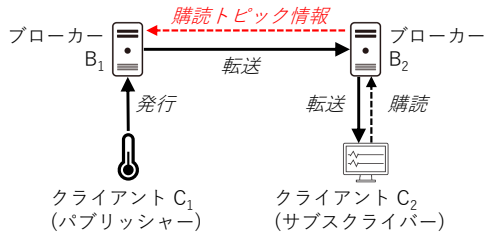


図 1: ブローカー間のトピック情報共有の例

Fig. 1　Example of sharing subscriptions among brokers

ばれる．パブリッシャーはメッセージにトピックを付与してブローカーへ送信し，サブスクライバーはブローカーに対して関心のあるトピックをあらかじめ指定して購読する．ブローカーでは，パブリッシャーから発行されたメッセージのトピックを参照し，当該トピックを購読しているサブスクライバーへメッセージの転送を行う．

MQTT ブローカーは全てのメッセージを仲介するため，大規模システムにおいては複数ブローカーを用いた負荷分散が必要となる [3]．これにより，ブローカー（群）に接続可能なクライアント数の増加や，クライアント間のメッセージのスループット向上が期待できる．こうした負荷分散においては，ブローカー間でトピック情報を共有することで，連携のオーバヘッドを低減し，スループットを向上可能であることが明らかとなっている [4], [5]．図 1 に例を示す．クライアント C_2 はブローカー B_2 に接続しており，何らかのトピックを購読している．ここで，ブローカー B_2 からブローカー B_1 へ当該トピックの情報を共有することで，ブローカー B_1 はクライアント C_1 が発行したメッセージをブローカー B_2 へ転送することの要否を判断することができる．このように，各ブローカーが配下のサブスクライバーの購読トピックを他ブローカーへ共有することで，ブローカー間のメッセージ転送量を抑えることが可能となる．

ブローカー間で共有するトピック情報は膨大な量となる可能性があるため，Bloom filter [6] を用いてコンパクトにトピック情報を保持する手法が提案されている [7]〜[9]．Bloom filter を用いることで，トピック共有に関するブローカー間の通信量を大幅に抑えつつ，共有先においてトピックの有無を検索することが可能となり，かつ省メモリ性をも得られる．しかしながら，既存手法では，ワイルドカードが含まれるトピックの扱いが問題となり得る．MQTT ではサブスクライバーによるトピック指定においてワイルドカードを含めることが可能であるが，Bloom filter はハッシュ値に基づく完全一致検索を想定したデータ構造であるため，ワイルドカードを含むトピックに対するマッチングを直接判定することはできない．このため，特にトピックに多様なワイルドカードが含まれる場合に，トピック検索時の Bloom filter に対するクエリ数が増大し，ブローカーのメッセージ処理性能に悪影響を及ぼすおそれがある．

本研究では，Bloom filter によるトピック情報管理において，ワイルドカードを効率的に扱う手法を提案する．提案手法ではトピックの接頭部を Bloom filter に登録し，検索時には存在し得るワイルドカードのパターンを考慮しつつ Trie [10] と類似した方式でクエリを処理する．これにより，トピックに多様なワイルドカードが含まれる場合であっても，トピック検索時の Bloom filter に対するクエリ数を小さく抑えることが可能となる．

以降では，まず 2. 章にて本研究における分散 MQTT ブローカーの想定を述べ，関連技術として MQTT トピックに関するプロトコル仕様と Bloom filter の概要を紹介する．続いて 3. 章では MQTT の購読トピック管理に関する関連研究を示す．その後，4. 章にて提案手法を述べ，5. 章でシミュレーション実験による評価について説明する．最後に 6. 章にて，まとめと今後の課題を述べる．

2. 分散 MQTT ブローカー

大規模システムにおいて MQTT を用いる場合，スループット向上やメッセージ配送遅延低減のために，複数ブローカーを用いた負荷分散が必要となる．本稿では，複数のブローカーが存在し，各クライアントはいずれか一つのブローカーに接続する状況を想定する．分散ブローカーには，透過性が求められる．すなわち，クライアントは，単一のブローカーを用いる場合と同様に，メッセージの発行や購読を行えなければならない．ここで，メッセージの発行とはブローカーに対する PUBLISH メッセージの送信とそれに付随する処理を指し，購読はブローカーに対する SUBSCRIBE メッセージの送信とそれに付随する処理を指す．詳細は MQTT のプロトコル仕様 [11] を参照されたい．

以降の議論において，以下の表記を用いる．

- 発行トピック：パブリッシャーが PUBLISH メッセージに付与するトピック
- 購読トピック：サブスクライバーが SUBSCRIBE メッセージで指定するトピック
- ローカルサブスクライバー：あるブローカーから見て，自身に直接接続しているサブスクライバー
- リモートブローカー：あるブローカーから見て，自身以外のブローカー
- リモートサブスクライバー：リモートブローカー配下のサブスクライバー

リモートサブスクライバーにも PUBLISH メッセージを届けるため，各ブローカーは，パブリッシャーからの PUBLISH メッセージを必要に応じてリモートブローカーへ転送する必要がある．単純な方法としては，全ての PUBLISH メッセージをブローカー間で相互に転送することで，リモートサブスクライバーへもメッセージを届けることができる [3]．しかしながら，この方式ではリモートサブスクライバーが必要としていないメッセージも転送することとなり，オーバヘッドが大きい．これに対し，図 1 で示したように，ブローカー間で購読トピックを共有することで分散ブローカー全体のスループットを向上可能であることが知られている [4], [5]．

本研究では，このブローカー間のトピック共有で用いるデータ構造に着目する．ブローカー間のトピック共有について，本

稿では以下の想定を置く．

- 各ブローカーは，ローカルサブスクライバーによる購読トピックをリモートブローカーに共有する．共有の宛先やタイミングについては本稿では限定しない．一例としては，直接接続している全てのリモートブローカーに対し，SUBSCRIBE メッセージの受信時や新規ブローカーの検出時に共有を行うことが考えられる．
- 各ブローカーは，PUBLISH メッセージ受信時，発行トピックがリモートブローカーから共有されている購読トピックに含まれているか否かを確認する．その結果に応じて，リモートブローカーへの当該 PUBLISH メッセージ転送等の処理を行う．

以上の想定は，ここまでに言及した既存研究を含め，多くの分散ブローカーに当てはまる．ブローカー間のネットワークトポロジーやルーティングについては様々なバリエーションが存在するが，それらの選択については本稿ではスコープ外とする．

購読トピックの集合は膨大な量となり得ることから，Bloom filter によってコンパクトに保持するアプローチが存在する [7]〜[9]．ブローカーはローカルサブスクライバーの購読トピックを Bloom filter に格納し，当該 Bloom filter をリモートブローカーへ共有する．PUBLISH メッセージの受信時は，その発行トピックが Bloom filter に含まれるか否かを確認し，その結果に応じてリモートブローカーへの転送等を行う．Bloom filter に対する検索結果は偽陽性を含むことから，実際にはサブスクライバーが存在しないトピックであっても，PUBLISH メッセージが転送される場合がある．この場合は，受信側のブローカーで破棄すれば良い．偽陽性の発生率は，Bloom filter のサイズを適切に設計することで小さく抑えることができる．購読の解除，すなわち UNSUBSCRIBE メッセージの処理については，ここでは詳細は割愛するが，Counting Bloom filter [12] 等の Bloom filter の派生データ構造を用いることで実現可能である．

なお，本稿の提案手法を含め，Bloom filter によるトピック管理は単体ブローカーにおいても適用可能である．ただし，前述のとおり偽陽性が含まれることから，各クライアントは不要なメッセージを破棄するというプロトコル仕様外の動作を求められる点には留意が必要である．

以降の節にて，MQTT におけるトピック仕様と，Bloom filter の詳細を述べる．

2.1 MQTT トピック

MQTT プロトコル [11] において，トピックはトピック階層をスラッシュ区切りで並べた文字列として規定されている．例えば，“building3/room2/temperature” は 3 階層から成るトピックである．トピックの長さは最大 65,535 バイトと定められている．

購読トピックにおいてのみ，2 種類のワイルドカードを用いることができる．一つ目は単一階層ワイルドカード “+” であり，二つ目は複数階層ワイルドカード “#” である．複数階層ワイルドカードは，任意の数のトピック階層とマッチするものであり，トピックの末尾にのみ用いることができる．これら 2 種類のワイルドカードは，組み合わせて使うことが可能である．

表 1: ワイルドカードトピックの例
Table 1　Examples of MQTT topics with wildcards

購読トピック	マッチする発行トピック
building3/+/temperature	building3/room1/temperature building3/room2/temperature
building3/#	building3/room1 building3/room2/humidity
+/#	building1/room1 building2/room5/temperature

表 1 にワイルドカードを用いた購読トピックの例を示す．

用途によっては，トピック数は極めて多くなり得る．一例として，空間 ID [13] をトピックに用い，位置に応じた情報のやり取りを行うケースが考えられる．空間 ID は Slippy map tilenames(注1) と類似した，実世界の空間に対して識別子を付与するスキームである．3 次元空間を階層的にボクセルに区切り，各ボクセルに対して一意な番号を割り当てることができる．標準的な設定では，最小粒度のボクセルは縦横 0.5m および高さ 0.6m であり，高層ビル 1 棟で百万超のボクセルを含む可能性がある．こうしたボクセルの ID をトピックに用いることを考えた場合，トピック数は膨大となり得ることから，トピック集合をコンパクトに扱う手法が必要となる．

2.2 Bloom Filter

Bloom filter [6] は要素の存在チェックを可能とする確率的データ構造である．長さ b のビット列であり，初期状態では全ビットが 0 となっている．要素の登録時は，まず k 個のハッシュ関数を用いて要素のハッシュ値を計算する．各ハッシュ値から，ビット列上の対応する位置を求め，当該ビットを 1 とすることで，要素の登録が完了する．要素の検索時は，k 個のハッシュ関数を用いて要素のハッシュ値を計算し，各ハッシュ値と対応するビット列上の位置を見る．全てのビットが 1 となっていれば，当該要素が高確率で存在すると判断できる．一つでも 0 のビットがあれば，当該要素は存在しない．なお，ビット論理和を計算することで，複数の Bloom filter をマージすることが可能である．

検索結果は，偽陽性を含み得る．偽陽性の発生率 p_{fp} は次式で計算できる．

$$p_{\mathrm{fp}} = (1 - e^{-\frac{kn_{\mathrm{e}}}{b}})^k$$

ここで，n_{e} は要素数である．p_{fp} を最小とする k は，次式で求められる．

$$k = \frac{b}{n_{\mathrm{e}}} \ln 2$$

上記の k に基づくと，b と p_{fp} の関係は次式で表すことができる．

$$b = -\frac{\ln p_{\mathrm{fp}}}{(\ln 2)^2} n_{\mathrm{e}}$$

(注1)：https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames (2024 年 8 月 1 日参照)

上式に基づき，許容し得る偽陽性発生率から Bloom filter の適切なサイズを求めることができる．

3. 関 連 研 究

Bloom filter は時間効率・空間効率に優れることから，トピック管理への適用が行われてきた．Dominguez ら [9], [14] は，Bloom filter の派生である Counting Bloom filter [12] をトピックベースの Publish/Subscribe 型メッセージングに適用し，MQTT プロトコルを用いて検証実験を行っている．MQTT 以外にも適用例は存在し，例えば代表的なメッセージ指向ミドルウェア RabbitMQ では Stream Filtering 機能の内部で Bloom filter が購読情報の管理に用いられている [15]．また，Content-Centric Networking および Named Data Networking においても Bloom filter をメッセージのルーティングに応用する研究が為されている [16], [17]．これらは要素の接頭部を Bloom filter に登録するという点で，本研究の提案手法と類似している．しかしながら，以上に挙げたいずれにおいても，ワイルドカードの効率的な処理は考慮されていない．

Naaman [7] および Chen ら [8] は，複数の MQTT ブローカ間における購読トピック共有に Bloom filter を導入している．ワイルドカードを考慮しており，ブローカ間で Bloom filter を共有することに加え，ワイルドカードトピックのパターンも共有する手法を提案している．ここでのパターンとは，2 種類のワイルドカードが出現する階層の情報を指す．例えば，あるブローカーが購読トピックとして “a/+/c”，“x/+/z”，“a/+/+”，“+/+/#” を持っている状況を考える．この場合，ワイルドカードトピックのパターンは $[\{1;-1\},\{1,2;-1\},\{0,1;2\}]$ のようになり，これが Bloom filter と合わせてリモートブローカーへ共有される．$\{0,1;2\}$ は第 0 階層と第 1 階層に単一階層ワイルドカードがあり，第 2 階層に複数階層ワイルドカードがあるような購読トピックが少なくとも一つ存在することを意味する．なお，パターンに含まれる −1 は，当該ワイルドカードが出現しないことを表している．ブローカーが PUBLISH メッセージを受信した際は，発行トピックに加え，ワイルドカードトピックのパターンを適用したバリエーション全てを，Bloom filter で検索する．一例として，発行トピックが “x/y/z” であり，ワイルドカードトピックのパターンが前述の例と同様であった場合，Bloom filter に対して “x/y/z”，“x/+/z”，“x/+/+”，“+/+/#” の 4 トピックを検索することとなる．この手法ではワイルドカードトピックのパターンが限定的であることを想定しており，多様なワイルドカードトピックが存在する場合には Bloom filter への検索クエリ数が肥大化するおそれがある．

4. 提 案 手 法

本研究では，Bloom filter を用いてワイルドカードトピックを効率的に扱う手法を提案する．提案手法では，購読トピックの接頭部を Bloom filter へ登録する．これにより，トピックの存在チェックをトピックツリー [18] や Trie [10] と類似した流れで行うことが可能となる．

2. 章で述べたように，本研究では，各ブローカーがローカル

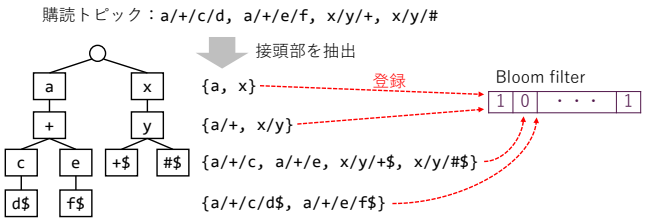


図 2: Bloom filter へのトピック接頭部の登録

Fig. 2　Adding topic prefixes to Bloom filter

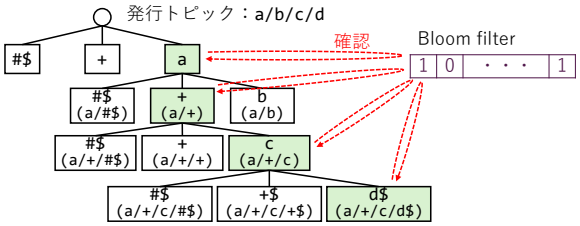


図 3: 発行トピックの検索

Fig. 3　Searching for published topic

サブスクライバーによる購読トピックをリモートブローカーに共有することを想定している．ブローカーが PUBLISH メッセージを受信した際は，リモートブローカーから共有された Bloom filter に対し，発行トピックを検索し，その結果に応じて当該メッセージの転送等を行う．以降の節では，提案手法におけるトピック管理の詳細について述べる．

4.1 Bloom filter への登録

トピックを Bloom filter へ登録するにあたり，提案手法ではトピックを複数の接頭部に分解する．ここで，トピックの接頭部とは，トピックの先頭から一定の深さまでのトピック階層を切り出したものである．例えば，トピック “foo/bar/baz” に対しては，“foo”，“foo/bar”，“foo/bar/baz” という三つの接頭部が考えられる．

Bloom filter へ登録する前処理として，末尾のトピック階層に終端記号を付与する．本稿では $ を終端記号として用いる．その上で，全ての接頭部を Bloom filter へ登録する．図 2 に例を示す．購読トピック “a/+/c/d” は，“a”，“a/+”，“a/+/c”，“a/+/c/d$” という四つの接頭部に分解され，それぞれが Bloom filter へと登録される．

ここで，Bloom filter のサイズは偽陽性を考慮し適切に定める必要がある．2.2 節に述べたように，許容可能な偽陽性発生率 p_{fp} と，想定される最大要素数 n_{e} から，ビット列の長さ b を計算可能である．また，b が求まれば，ハッシュ関数の個数 k についても 2.2 節の式から求めることができる．例えば，$p_{\mathrm{fp}} = 0.001$ および $n_{\mathrm{e}} = 100{,}000$ である場合，b は約 1,437,759 ビットとなり，k は約 10 となる．

4.2 トピックの検索

発行トピック t_p の検索は，図 3 に示すように，ワイルドカードのパターンを考慮しつつ Trie と類似した形で行う．ここで，t_p の階層数を d とし，$l_0, l_1, \ldots, l_{d-1}$ は t_p の各トピック階層を表すものとする．

まず，Bloom filter に対して階層数 1 の接頭部 “#$”，“+”，$l_0$

を検索する．t_p にマッチする購読トピックが存在する場合に，階層数 1 の接頭部として存在し得るものはこれら 3 種類のみである．従って，これらのいずれも Bloom filter に存在しなかった場合，マッチする購読トピックは存在しないと判断できる．一方，いずれかが存在していた場合には，階層数を増やして検索を継続する．この時，存在していた接頭部に対して “#$”，“+”，$l_1$ を付加した階層数 2 の接頭部 3 種類をそれぞれ Bloom filter に対して検索する．以降，検索する接頭部の階層数が t_p の階層数 d と一致するまで，同様の処理を繰り返す．なお，階層数 d の接頭部を検索する際には，末尾が “+” および l_{d-1} である接頭部には終端記号 “$” を付加してから検索する．以上の検索プロセスにおいて，終端記号 “$” を伴ういずれかの接頭部が Bloom filter に存在していた場合は，その時点で検索を終了し，マッチする購読トピックが存在するものと判断する．最終的に終端記号 “$” を伴う接頭部が Bloom filter に存在しなかった場合は，マッチする購読トピックは存在しないものと判断する．

図 3 は，図 2 で例示した Bloom filter に対する検索を行う場合の例を示している．Bloom filter に存在する接頭部は色付きのノードで表されている．この例では，接頭部 “a”，“a/+”，“a/+/c” が Bloom filter に存在しており，最終的に “a/+/c/d$” もヒットすることで，マッチする購読トピックが存在すると判断される．

検索プロセスにおいては，以下の点に留意が必要である．

- 深さ優先探索を行う．
- 各階層数において，末尾が “#$” である接頭部を優先的に検索する．

これらを考慮することで，マッチする購読トピックが存在する場合に，早期に検索プロセスを終了することが可能となる．

Algorithm 1 に擬似コードを示す．SEARCH 関数は，*topic* が高い確率で存在する場合に **true** を返し，確実に存在しない場合に **false** を返す．PARSE_TOPIC は，トピックをトピック階層に分割し，それらのリストを返す関数である．また，CONCAT は引数に与えられた複数の文字列を連結する関数である．SEARCH_HELPER 関数は，深さ優先探索を行うための再帰関数となっている．引数の $prefix$ は，一つ前の階層数において Bloom filter に存在していた接頭部を表す．$level$ は検索を行う接頭部の階層数であり，$levelList$ は発行トピックのトピック階層のリストである．8 行目から 14 行目にかけて，検索する階層数 $level$ の接頭部を用意している．15 行目から 23 行目では，階層数 $level$ を増加させながら再帰的に検索を行っている．16 行目の CHECK_BF は，引数に与えられた要素が Bloom filter に存在するか否かを確認する関数である．

Algorithm 1 検索プロセス

```
 1: function SEARCH(topic)
 2:     levelList ← PARSE_TOPIC(topic)
 3:     d ← levelList.LENGTH( )
 4:     levelList[d − 1] ← CONCAT(levelList[d − 1], “$”)
 5:     return SEARCH_HELPER(“”, 0, levelList)
 6: end function

 7: function SEARCH_HELPER(prefix, level, levelList)
 8:     pList[0] ← CONCAT(prefix, “/#$”)
 9:     if level = levelList.LENGTH( ) −1 then
10:         pList[1] ← CONCAT(prefix, “/+$”)
11:     else
12:         pList[1] ← CONCAT(prefix, “/+”)
13:     end if
14:     pList[2] ← CONCAT(prefix, “/”, levelList[level])
15:     for i ← 0 to 2 do
16:         if CHECK_BF(pList[i]) = true then
17:             if pList[i].ENDS_WITH(“$”) = true then
18:                 return true
19:             else
20:                 return SEARCH_HELPER(
                                pList[i], level + 1, levelList)
21:             end if
22:         end if
23:     end for
24:     return false
25: end function
```

4.3 計 算 量

提案手法では，いずれかの階層数において，考え得る接頭部がいずれも Bloom filter に存在しない場合，検索が打ち切られる．これにより，ワイルドカードのパターンを網羅的に検索する既存手法と比べ，Bloom filter に対する検索クエリ数を低減できると考えられる．

表 2 は時間計算量および空間計算量を比較したものである．“BF” は存在し得るワイルドカードのパターンを全て検索する方式であり，ベースラインとして比較対象としている．“BF w/ patterns (BFP)” は 3. 章で述べた Naaman らの手法 [7], [8] であり，以降ではこれを BFP と表記する．また，以降において以下の各表記を用いる．

表 2: 計算量の比較
Table 2 Asymptotic analysis

	BF	BF w/ patterns (BFP)	Proposed
空間計算量	$O(n_\mathrm{t})$	$O(n_\mathrm{t} + n_\mathrm{p} n_\mathrm{w})$	$O(n_\mathrm{t} n_\mathrm{lv})$
時間計算量（平均）	ワイルドカードの使われ方に依存		
時間計算量（最悪）	$O(2^{n_\mathrm{lv}})$	$O(2^{n_\mathrm{lv}})$	$O(2^{n_\mathrm{lv}})$

- n_t：購読トピック数
- n_lv：トピックの平均階層数
- n_p：ワイルドカードのパターン数
- n_w：一つのワイルドカードパターンにおいて用いられる平均ワイルドカード数

空間計算量については，提案手法ではトピックの接頭部を Bloom filter に登録するため，BF や BFP と比べ，より大きなサイズの Bloom filter が必要となる．2.2 節で述べたように，Bloom filter の適切なサイズは要素数に比例するため，提案手法においては $O(n_\mathrm{t} n_\mathrm{lv})$ となる．ただし，Bloom filter 自体が極めてコンパクトなデータ構造であり，1 要素あたり 10 ビット前後で格納が可能であることから，実際的な影響は限定的であると

表 3: シミュレーションのパラメーター
Table 3　Simulation parameters

パラメーター	デフォルト値
購読トピック数 n_t	6,000
トピック階層数 n_{lv}	6
ワイルドカード確率 p_{wc}	0.5
最小ワイルドカード階層 l_{wc}	0

考えられる．平均時間計算量については，ワイルドカードの用いられ方に強く依存する．このため，5. 章にてシミュレーション実験を通して手法間の比較を行う．最悪時間計算量は，いずれの手法も指数オーダーである．しかしながら，提案手法では最悪ケースは発生しづらいと考えられる．即ち，図 3 に例示したような，Bloom filter に埋め込まれたツリー構造に対して，途中で打ち切られることなく全ての分岐を最大階層数まで検索するケースは，容易には発生しない．対照的に，既存手法においては最悪ケースやそれに近い状況が比較的容易に発生し得る．これは，マッチする購読トピックが存在しない場合に，考え得るパターンを全て検索しなければ検索プロセスを終了できないためである．これらについても，5. 章において定量的な議論を行う．

5. 評　　価

提案手法の特徴を明らかにするため，Java 言語にて実装したシミュレーターを用いて実験を行った．評価指標は，発行トピックにマッチする購読トピックの有無を確認する際に必要となる，Bloom filter に対する平均検索クエリ数である．比較対象は 4. 3 節に述べた BF および BFP とする．

シミュレーションのパラメーターを表 3 に示す．各パラメーターについて，他のパラメーターをデフォルト値に固定した状態で値を変化させ，実験を行った．n_t は Bloom filter に格納する購読トピック数を表す．各購読トピックのトピック階層は 10 文字のランダムな文字列あるいはワイルドカードである．n_{lv} は各トピックの階層数を意味する．実験では，分析を容易とするため，全トピックの階層数を同じ値としている．p_{wc} は，n_t 個の購読トピックにおいて，ワイルドカードを含むものが存在する確率を表している．例えば $p_{wc} = 0.5$ の場合，期待値としては $n_t/2$ 個のトピックにワイルドカードが含まれることになる．なお，個々のワイルドカードトピックは以下のように生成している．まず，ワイルドカードを含まないトピックを一つ生成する．次に，そのトピックをもとに，考え得る全てのワイルドカードトピックを生成する．そして，それらの中から等確率でいずれか一つを選ぶ形で，一つのワイルドカードトピックが生成される．l_{wc} は，ワイルドカードが現れる最小階層である．例えば $l_{wc} = 0$ の場合，どの階層もワイルドカードとなり得る．

実験は，以下 3 パターンの方法で購読トピックを用意して実施した．

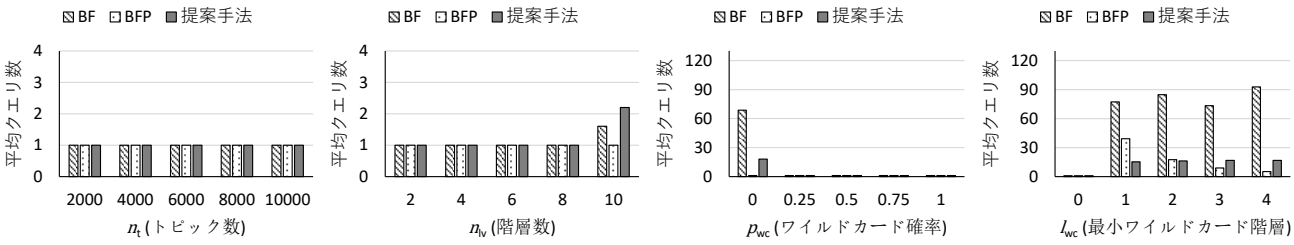
- ワイルドカードのみの購読トピックを許容（“#”，“+/#” 等）
- ワイルドカードのみの購読トピックを除外
- ワイルドカードのみの購読トピックを除外し，かつ発行トピックとマッチする購読トピックを除外

ワイルドカードのみのトピックは，多くの発行トピックとマッチする．特に “#” は全トピックとマッチするため，サブスクライバーに対する通信量が膨大となり得る．このため，大規模システムにおいては，クライアントはこうしたワイルドカードのみのトピックの利用を避けることが考えられる．また，MQTT のような Publish/Subscribe 型プロトコルはクライアント間の疎結合性を実現するものであり，ある PUBLISH メッセージの購読トピックが常に存在するとは限らない．こうした様々な状況をそれぞれ評価するために，上記のパターン分けを行っている．なお，発行トピックの階層数は購読トピックと同じく n_{lv} とし，各トピック階層は 10 文字のランダムな文字列としている．

図 4 から図 6 にシミュレーション結果を示す．実験はパターンごとに 10 回試行し，Bloom filter に対する検索クエリ数の平均値を算出している．これらの実験結果より，全体として，提案手法は既存手法と比べ少ない検索クエリ数となっていることがわかる．特に，トピック階層数の増大に対しては，提案手法の検索クエリ数が大幅に少なくなっている．図 6(b) の $n_{lv} = 10$ のケースでは，提案手法の平均検索クエリ数は 29.7 であり，BFP の平均検索クエリ数 1, 570.0 の 2 パーセント未満となっている．

図 4(a)，図 4(b) および図 4(c) では，多くのパターンにおいて，平均検索クエリ数は 1 となっている．これは，“#” のような，ほとんどの発行トピックとマッチする購読トピックが高確率で生成されているためと考えられる．いずれの手法においても，このようなトピックは優先的に検索することで検索プロセスを早期に終了させることができる．図 4(c) の $p_{wc} = 0$ においては提案手法と BF の平均検索クエリ数がやや大きくなっている．これは上記のような検索プロセスの早期終了をもたらすワイルドカードトピックが存在しないためと考えられる．BFP の場合は，ワイルドカードトピックが存在しなければ，ブローカー間で共有されるワイルドカードトピックパターンも空となるため，平均検索クエリ数は 1 に抑えられている．図 4(d) の $l_{wc} \geq 1$ のケースでは，全発行トピックにマッチする “#” が除外されるため，どの手法においても平均検索クエリ数は 1 を超える値となっている．BFP の場合，l_{wc} が大きくなるほど，ワイルドカードトピックのパターンが減るため，平均検索クエリ数も減少する傾向がある．

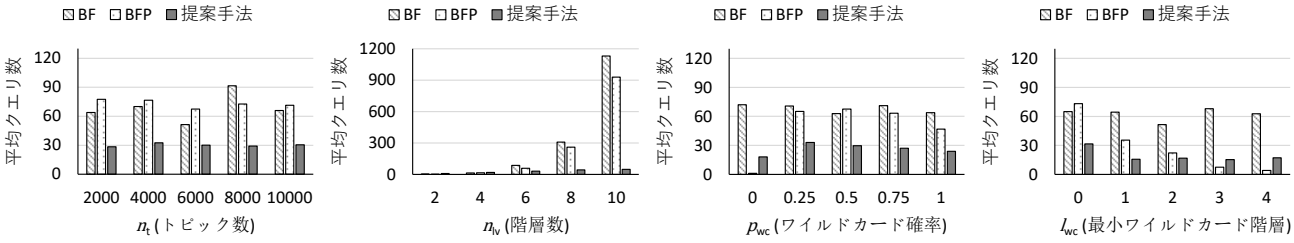
図 5(a)，図 5(c) および図 5(d) では，BF の平均検索クエリ数はどのパターンにおいても概ね同程度となっている．BF では，発行トピックから考え得るワイルドカードトピックのパターン全てを検索するが，n_t，p_{wc}，l_{wc} はこのパターン数に影響しないため，このような結果となっている．一方，図 5(b) に示されているように，n_{lv} はパターン数に大きく影響するため，平均検索クエリ数が大きく変動している．BFP の平均検索クエリ数は，主に購読トピックのワイルドカードトピックパターン数に依存する．n_{lv} が増えるとパターン数が増えて平均検索クエリ数も増大し，対照的に l_{wc} が増えるとパターン数が減少することで平均検索クエリ数も少なくなっている．



(a) (b) (c) (d)

図 4: 平均検索クエリ数（ワイルドカードのみのトピックを含む場合）

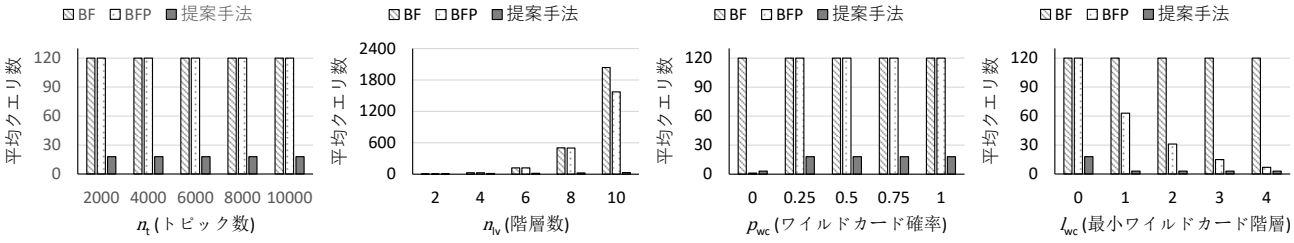
Fig. 4 Average number of queries (with wildcard-only topics)



(a) (b) (c) (d)

図 5: 平均検索クエリ数（ワイルドカードのみのトピックを含まない場合）

Fig. 5 Average number of queries (without wildcard-only topics)



(a) (b) (c) (d)

図 6: 平均検索クエリ数（マッチする購読トピックを含まない場合）

Fig. 6 Average number of queries (without matching subscription)

図 6 は，発行トピックにマッチする購読トピックが存在しないケースの実験結果を表している．こうしたケースでは，BF および BFP はそれぞれ考え得るパターンを網羅的に検索することとなり，平均検索クエリ数が増大する．一方，提案手法では，ある階層数において考え得る接頭部がいずれも存在しなければ検索プロセスを打ち切ることができるため，既存手法と比べ平均検索クエリ数を少なく抑えることができている．

特に図 5(b) および図 6(b) では，顕著な差が生じており，提案手法は既存手法よりも大幅に少ない平均検索クエリ数を達成している．Bloom filter に対する検索が，PUBLISH メッセージの受信ごとに発生することを考えると，こうした検索クエリ数の差は分散ブローカーのスループット性能等に大きな影響を与える可能性がある．これについて，次節にてさらなる検証を行う．

5.1 ブローカー性能への影響

本節では，Bloom filter に対する検索クエリ数が MQTT ブローカーの性能に与える影響について検証する．提案手法は 2. 章に述べたように分散ブローカーを想定したものであるが，ここではブローカー性能への影響を明らかにするために単体ブローカーを対象として実験を行う．即ち，パブリッシャーとサブスクライバーの間を 1 台のブローカーが仲介する状況において，ブローカー内に Bloom filter への検索処理を組み込むことで，性能面に生じる影響を明らかにする．評価指標としては，平均送出スループットおよび平均遅延を用いる．平均送出スループットは，ブローカーからサブスクライバーへと単位時間あたりに送出されるメッセージ数の平均値であり，平均遅延はパブリッシャーからサブスクライバーへメッセージが届くまでに要する時間の平均値である．

ブローカーとしては HiveMQ CE 2024.5(注2) を用い，クライアントとして MQTTLoader v0.8.6(注3) を用いる．ブローカーは，PUBLISH メッセージの受信時に簡易的な Bloom filter の検索処理が行われるように改変した．Bloom filter の実装としては，guava 33.1.0-jre(注4) を用いている．なお，同ライブラリで用いら

(注2)：https://github.com/hivemq/hivemq-community-edition (2024 年 8 月 1 日参照)

(注3)：https://github.com/dist-sys/mqttloader (2024 年 8 月 1 日参照)

(注4)：https://guava.dev/ (2024 年 8 月 1 日参照)

表 4: マシンスペック
Table 4　Host machine specs

項目	内容
CPU	Intel Core i9-12900
メモリ	64 GB
ネットワーク	1 GbE
OS	Ubuntu 22.04

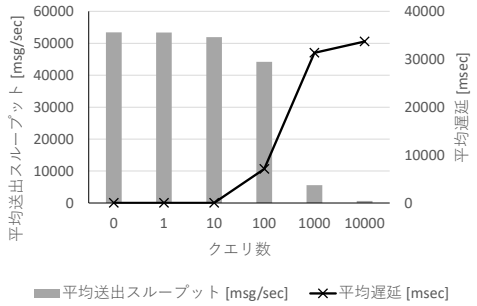


図 7: Bloom filter への検索がブローカー性能に与える影響
Fig. 7　Affect of searching Bloom filter on broker performance

れているハッシュ関数は 128 ビットの MurmurHash3 である．Bloom filter のパラメーターである格納要素数は 10,000 とし，偽陽性率は 0.001 とした．Bloom filter には，事前に長さ 100 のランダムな文字列 10,000 個を格納した．その上で，PUBLISH メッセージの受信時に，Bloom filter に対する検索クエリがあらかじめ指定した数だけ発生するように実装した．

MQTTLoader のパラメーターは以下のとおり設定した．

- MQTT プロトコルバージョン：v5.0
- パブリッシャー数およびサブスクライバー数：各 1
- ランプアップ時間およびランプダウン時間：各 5 秒
- 実行時間：70 秒
- ペイロードサイズ：1,024 バイト
- パブリッシャーの送信間隔：10 マイクロ秒

実験では，パブリッシャー、ブローカー、サブスクライバーとしてそれぞれ 1 台ずつのホストマシンを用いた．マシンスペックは表 4 のとおりである．

図 7 に結果を示す．検索クエリ数の増加に伴い，スループットは減少し，遅延は増大している．図 4 から図 6 に示した結果を考慮すると，平均検索クエリ数が 100 あるいは 1,000 を超えるケースがある既存手法と比べ，提案手法では高いスループットと低い遅延を期待できることがわかる．

6. ま と め

本稿では，分散 MQTT ブローカーを想定し，Bloom filter を用いてワイルドカードトピックを効率的に扱う手法を提案した．提案手法では，Bloom filter に対し購読トピックの接頭部を登録し，ワイルドカードのパターンを考慮しつつ Trie と類似した検索を行う．シミュレーション実験の結果より，提案手法が既存手法と比べ多くのケースにおいて Bloom filter に対する平均検索クエリ数を低減できることが明らかとなった．特にトピックの階層数が大きい場合に顕著であり，階層数が 10，かつマッチする購読トピックが存在しないケースでは，提案手法の平均検索クエリ数は既存手法の 2 パーセント未満であった．さらに，オープンソースのブローカー実装を用いた実機検証により，Bloom filter に対する検索クエリ数がブローカー性能に大きな影響を与える可能性が示された．

今後の課題として，ブローカー間で Bloom filter を共有するアルゴリズムの検討，および複数台のブローカーを用いた性能検証が挙げられる．

謝辞　本研究は JST さきがけ JPMJPR21P8 の支援を受けたものである．

文　献

[1] MQTT, https://mqtt.org/ (accessed Aug. 1, 2024).
[2] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/subscribe,” ACM Computing Surveys, vol.35, no.2, pp.114–131, 2003.
[3] E. Longo, A.E. Redondi, M. Cesana, A. Arcia-Moret, and P. Manzoni, “Mqtt-st: a spanning tree protocol for distributed mqtt brokers,” Proc. IEEE International Conference on Communications, pp.1–6, 2020.
[4] R. Banno, J. Sun, S. Takeuchi, and K. Shudo, “Interworking layer of distributed mqtt brokers,” IEICE Transactions on Information and Systems, vol.E102.D, no.12, pp.2281–2294, 2019.
[5] A. Detti, L. Funari, and N. Blefari-Melazzi, “Sub-linear scalability of mqtt clusters in topic-based publish-subscribe applications,” IEEE Transactions on Network and Service Management, vol.17, no.3, pp.1954–1968, 2020.
[6] B.H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol.13, no.7, pp.422–426, 1970.
[7] N. Naaman, “Large scale connectivity infrastructure for an internet of things platform,” Software Architecture Conference, 2016.
[8] C. Chen, B. Mandler, N. Naaman, and Y. Tock, “Publish-subscribe system with reduced data storage and transmission requirements,” U.S. Patent 9 886 513, 2018.
[9] A.M. Dominguez, R. Alcarria, E. Cedeno, and T. Robles, “An extended topic-based pub/sub broker for cooperative mobile services,” International Conference on Advanced Information Networking and Applications Workshops, pp.1313–1318, 2013.
[10] E. Fredkin, “Trie memory,” Commun. ACM, vol.3, no.9, pp.490–499, 1960.
[11] OASIS Standard, “MQTT Version 5.0,” https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed Aug. 1, 2024), 2019.
[12] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, “Summary cache: a scalable wide-area web cache sharing protocol,” IEEE/ACM Transactions on Networking, vol.8, no.3, pp.281–293, 2000.
[13] 独立行政法人情報処理推進機構，“4 次元時空間情報基盤 ガイドライン,” https://www.ipa.go.jp/digital/architecture/guidelines/4dspatio-temporal-guideline.html (accessed Aug. 1, 2024).
[14] A.M. Dominguez, T. Robles, R. Alcarria, and E. Cedeño, “A rendezvous mobile broker for pub/sub networks,” International Conference on Green Communications and Networking, pp.16–27, 2013.
[15] A. Cogoluègnes, “Stream filtering internals,” https://www.rabbitmq.com/blog/2023/10/24/stream-filtering-internals (accessed Aug. 1, 2024), 2023.
[16] J. Lee, M. Shim, and H. Lim, “Name prefix matching using bloom filter pre-searching for content centric network,” Journal of Network and Computer Applications, vol.65, pp.36–47, 2016.
[17] J. Kim, M.-C. Ko, J. Kim, and M.S. Shin, “Route prefix caching using bloom filters in named data networking,” Applied Sciences, vol.10, no.7, 2226, 2020.
[18] L. Brandl, “Mqtt topic tree & topic matching: Challenges and best practices explained,” https://www.hivemq.com/blog/mqtt-topic-tree-matching-challenges-best-practices-explained/ (accessed Aug. 1, 2024), 2023.