ShutPub: Publisher-side Filtering for Content-based Pub/Sub on the Edge

Minghe Wang
TU Berlin & ECDF
Berlin, Germany
mw@mcc.tu-berlin.de

Trever Schirmer
TU Berlin & ECDF
Berlin, Germany
ts@mcc.tu-berlin.de

Tobias Pfandzelter
TU Berlin & ECDF
Berlin, Germany
tp@mcc.tu-berlin.de

David Bermbach
TU Berlin & ECDF
Berlin, Germany
db@mcc.tu-berlin.de

Abstract

The pub/sub paradigm facilitates communication among heterogeneous edge and IoT devices for distributed edge applications. At the same time, the increasing number of devices and sensors at the edge leads to higher network congestion, which requires more processing power for both pub/sub brokers and devices. While message filtering based on subscriber-specified rules can alleviate this, implementing filtering logic on the brokers still requires publishing components to send all messages over constrained links.

In this paper, we instead propose filtering pub/sub messages directly on the publisher, in order to reduce network congestion. We present ShutPub, a publisher-side middleware that performs message filtering before forwarding them to the broker. ShutPub limits the publisher message dissemination based on subscriptions and their filters while still being transparent to the publisher. In this way, the content-based filter capabilities are moved from the broker to the publisher so that only messages that have a receiver are transmitted. Our prototype evaluation shows that ShutPub can reduce system strain on the network and broker without simply shifting the burden to the publisher, which benefits from sending fewer messages.

Keywords

Publish/Subscribe, Edge Computing, Publisher-Side Filtering

ACM Reference Format:
Minghe Wang, Trever Schirmer, Tobias Pfandzelter, and David Bermbach. 2024. ShutPub: Publisher-side Filtering for Content-based Pub/Sub on the Edge. In *7th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3642968.3654815

1 Introduction

By bringing computational power closer to the end user, edge computing enables real-time communication and privacy for users while also alleviating network strain. Often, applications running in such edge environments follow the publish/subscribe (pub/sub) paradigm as it can support decentralized, scalable, and asynchronous communication [7–10, 19]. Today, state-of-the-art pub/sub in research and practice usually relies on content-based filtering (often on topic-based pub/sub), where messages are distributed based on their content, as it allows clients to precisely express their interests and to only deliver those messages needed, thus, minimizing unnecessary resource utilization [12, 16–18, 20, 22].

Even in this case, however, subscribers may be only interested in a fraction of the messages published, with the pub/sub broker immediately discarding many messages upon receipt, e.g., in case of messages published to a topic that does not have subscribers. As a result, the publisher unnecessarily creates and sends messages to at least one broker, which in turn run(s) content-based filtering, thus, wasting resources. If instead parts of the content-based filtering were shifted to the publisher, we could avoid this inefficiency by discarding messages already there or possibly even not generating the messages in the first place.

In this paper, we propose ShutPub, a publisher-side middleware for shifting content-based filtering from broker to publisher without modifying publisher logic. Using ShutPub, publishers only send messages that are of interest to subscribers, limiting message distribution directly at the source and thus saving network bandwidth. The messages in ShutPub are still processed in real-time, on the shortest path from sender to recipient, but (i) content is filtered directly on the publisher, (ii) publishers *only* send messages to the broker when there is a subscriber that is interested in the current message content, and (iii) *any* broker can benefit from this, as message filtering is performed on the publisher.

Essentially, publishers receive filters from the broker based on existing content-based subscriptions. Then, the publishers filter out unnecessary messages accordingly. We make the following contributions:

- We describe the design of ShutPub, an edge publisher-side middleware that benefits *any* content-based pub/sub system (§3).
- We present a proof-of-concept ShutPub prototype (§4).
- We demonstrate ShutPub and compare its performance to existing content-based pub/sub brokers (§5).

2 Related Work

In edge environment, pub/sub provides an asynchronous and many-to-many way for communication among heterogeneous edge nodes [7–10, 19]. Content-based pub/sub, especially in the form of topic-based pub/sub, is typically used as it supports delivering messages only to those who are interested in them.

Current research uses different approaches to reduce network strain in the content-based pub/sub system. Some approaches use covering algorithms to deliver a message only to a specific subset of subscribers [2, 4, 12, 15, 22]. This leads to fewer subscribers that have to receive a given message. Other work uses content-based filtering on the broker using boolean expressions to enable more expressive subscriptions [1, 5, 6, 13], which leads to fewer messages that have to be sent to subscribers. Others improve inter-broker routing to reduce the overall number of messages exchanged, e.g., [8–10]. There is even work, integrating Function-as-a-Service (FaaS) to enable elastic, on-demand complex data processing on the broker [14, 21]. This allows offloading some message processing effort from subscribers to the broker, which can reduce both the number and size of messages.

However, all of these systems still require brokers to receive *all* messages from publishers and perform matching, scoring, filtering, or complex processing on *all* received messages. This can reduce the load between brokers and subscribers, but not between brokers and publishers. Performing publisher-side filtering can fill the research gap and further reduce resource consumption in edge environments, which have constrained network and computational resources.

There is some early research on message filtering on the publisher side: Cho et al. [3] proposed an approach that modifies CORBA event services and their counterparts on publishers and subscribers to partially filter events on the publisher side. A key limitation of this is that this requires tight coupling between publishers, brokers, and subscribers and is unlikely to scale beyond a low number of involved nodes. ShutPub, in contrast, tries to keep as much as possible unchanged: Subscribers are not affected, it integrates with any kind of broker, and brokers retain their functionality for unmodified publishers. In essence, ShutPub more resembles a side-car proxy with message filtering than a fully integrated publisher-to-subscriber message channel.

Hashemi et al. [11] propose to use publisher-side message filtering as a small part of a larger data sharing architecture. In contrast to our approach, their work focuses on security and access control aspects, trying to maintain data privacy through filtering on the publisher side, i.e., it filters data that *may not* leave the publisher, similar to information flow control. ShutPub, however, filters data that *may* leave the publisher and focuses on not sending events for which no subscriber exists to improve resource efficiency. Furthermore, their approach appears to not have been implemented.

3 ShutPub Architecture

With ShutPub, we aim to offload parts of the content-based filtering from the broker to the publishers, i.e., we separate the question of *whether* there are subscribers for a given message from the question of *who* those subscribers are. The first question is pushed to the publisher side while the second question is handled on the broker. In fact, it would even be possible to have a simple filter with occasional false positives on the publisher in case that the content-based filtering should prove very compute-intensive.

Overall, we design ShutPub with two main objectives: (i) enabling accurate event distribution while reducing network strain between publisher and broker, and (ii) optimizing content-based pub/sub systems by offloading filtering to the publisher. In this section, we give an overview of the architecture and design of ShutPub; see also Figure 1.

ShutPub consists of two main components: the *Publisher-Side Middleware* and the *Filter Manager*. The Publisher-Side Middleware is located on the publisher and the Filter Manager in the broker. Every ShutPub publisher consists of an original publisher without any changes (e.g., a sensor that publishes a reading in regular intervals (Step 1 in Figure 1)), and a Publisher-Side Middleware instance that filters these messages before actually sending them to the broker.

In detail, the Publisher-Side Middleware subscribes to its own meta topic (Step 2 in Figure 1) to receive the filter it should use for filtering messages (Step 6 in Figure 1). The Filter Manager is responsible for generating, maintaining and publishing these filters based on the incoming subscriptions. If there is a new subscriber connection on a topic with

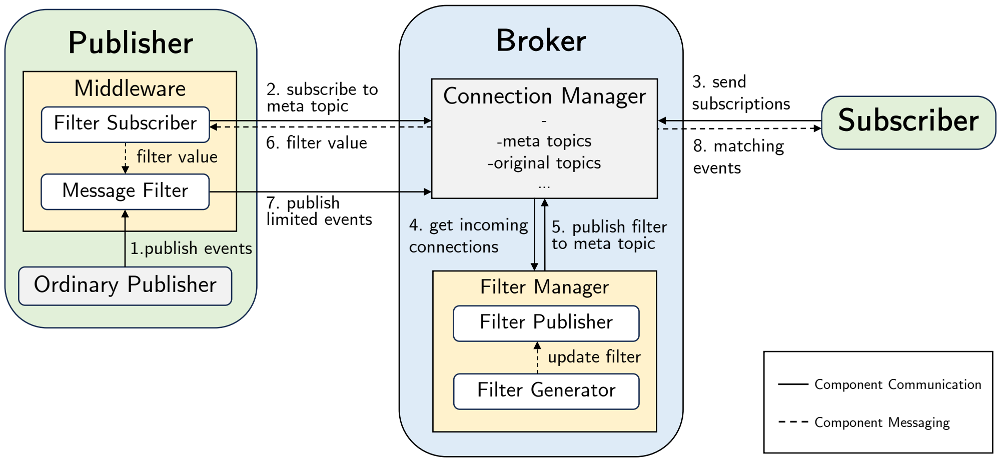


Figure 1: The main component of ShutPub is a Publisher-Side Middleware intended to reduce network strain between publisher and broker by filtering messages directly on the publishers. On the broker side, a Filter Manager is responsible for generating filters based on subscriptions, which are then forwarded to the Publisher-Side Middleware via a meta topic.

existing subscribers (Step 3 in Figure 1), the Filter Manager will analyze all existing subscriptions and create a "superfilter", i.e., a filter that is a superset of all existing filters (Step 4 in Figure 1). It then publishes this filter to the meta topic (Step 5 in Figure 1). The Publisher-Side Middleware will only publish a message if it matches this superfilter (Step 7 in Figure 1). In this way, ShutPub pre-processes messages on the publisher side for the content-based pub/sub broker which optimizes the overall system by transmitting and processing fewer messages.

Overall, ShutPub uses pub/sub basic functionality to optimize current pub/sub system, and (i) reduces the network strain between publishers and the broker, and (ii) optimizes *any* content-based pub/sub broker by restricting publisher message dissemination.

4 Implementation

To show the feasibility of the proposed system design, we implemented a proof-of-concept prototype of ShutPub that we have made available as open-source software.[1] The prototype is based on Apache ActiveMQ Artemis[2] which is an open-source, scalable, well-documented pub/sub system. ShutPub extends ActiveMQ publishers and brokers. For the broker, ShutPub extends it with a key-value filter map that matches subscription filters to topics. The filter map stores the superset of multiple filters on the same topic, which is received from the broker. In this way, ShutPub avoids adding complex computation to the publisher, and the ActiveMQ content-based filtering functionality can ensure accurate message dissemination to the subscribers. Once a publisher connects to the broker with a specific topic, it will receive the corresponding filter over a meta topic. For the publisher, ShutPub lets every publisher initialize a new Publisher-Side Middleware instance that subscribes to the meta topic for filter values, and performs message filtering based on that. Essentially, ShutPub uses pub/sub functionality to add new features to pub/sub system.

5 Evaluation

In this section, we evaluate ShutPub by running experiments. We run all ShutPub experiments on Google Cloud in the `europe-west3` region and use three `e2-standard-2` (2 vCPUs, 8GB RAM) machines, one each for the broker, publishers, and subscribers. For overheads, we measure CPU utilization as the majority of extra efforts resulting from the use of ShutPub are CPU-bound. As a baseline, we compare ShutPub to ActiveMQ without any changes.

We start by quantifying network load effects (§5.1). Afterwards, we evaluate overheads on broker and publishers to demonstrate the impact of ShutPub. For this, we first measure overheads on the publisher arising from an increased message sending frequency (§5.2) before studying overheads on the broker by respectively increasing throughput from the publishers (§5.3) and filter update frequency from subscribers (§5.4). Finally, we measure the overhead on the broker from maintaining more filters (§5.5) and discuss our findings (§5.6).

5.1 Network Load Effects

ShutPub allows publishers to filter messages before sending them to the broker thus reducing the publisher-side network

[1] https://github.com/Mhwwww/ShutPub
[2] https://github.com/apache/activemq-artemis

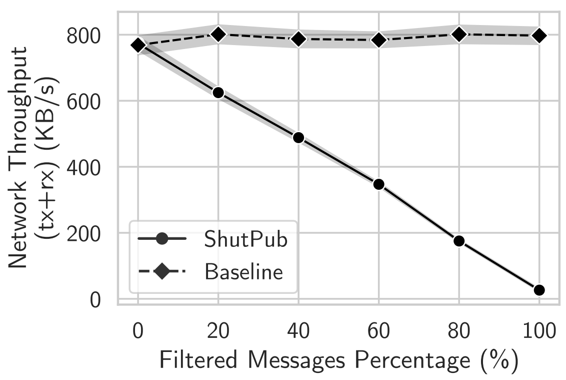


Figure 2: As the proportion of filterable messages increases, the ShutPub network load decreases to near zero. The baseline network throughput, however, stays around 800 kBps per second as it always needs to send all messages to the broker.

load. To evaluate how ShutPub impacts network bandwidth between publishers and the broker, we let 1,000 publishers connect to one broker and set ActiveMQ publishers to be the baseline. We vary the percentage of messages that can be filtered by publishers from 0 to 100 percent in twenty percent increments. To measure the network load between publishers and the broker, we collect the receiving and transmitting network throughput of publishers. The result is shown in Figure 2.

As the percentage of filtered messages increases, the network throughput of ShutPub decreases from 773.75 to 26.35 kBps (kilobytes per second), while the baseline network throughput stays at a stable level of 789.90 kBps on average. As ShutPub needs to maintain meta-topics, it introduces additional overhead when no messages need to be filtered out. We measured a 5.11 kBps communication overhead with 1,000 publishers connected to ShutPub, which is compensated by the reduction of publisher messages if only 0.56% of messages are filtered.

5.2 Publisher Overheads

To evaluate how ShutPub impacts publisher performance, we vary the publisher message sending frequency from 10,000 to 40,000 messages per second in steps of 10,000 with one out of every ten messages matching the filter. We compare it to a scenario using the vanilla ActiveMQ publisher and then measure the CPU utilization of the publishers. The results are shown in Figure 3.

As the publisher sends more messages, the CPU utilization increases for both ShutPub and baseline. The average baseline CPU utilization is 42.18 pp (percentage points) higher than ShutPub, although ShutPub needs to maintain filters

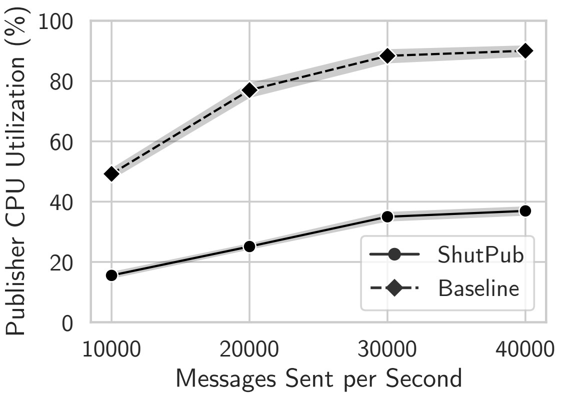


Figure 3: As the publisher sends messages more frequently, the CPU utilization increases for both baseline and ShutPub.

and perform message filtering on the publisher. This is because filtering messages on the publisher consumes fewer resources than sending messages to the broker, i.e., the benefits exceed the measurable costs.

5.3 Broker Overheads: Message Frequency

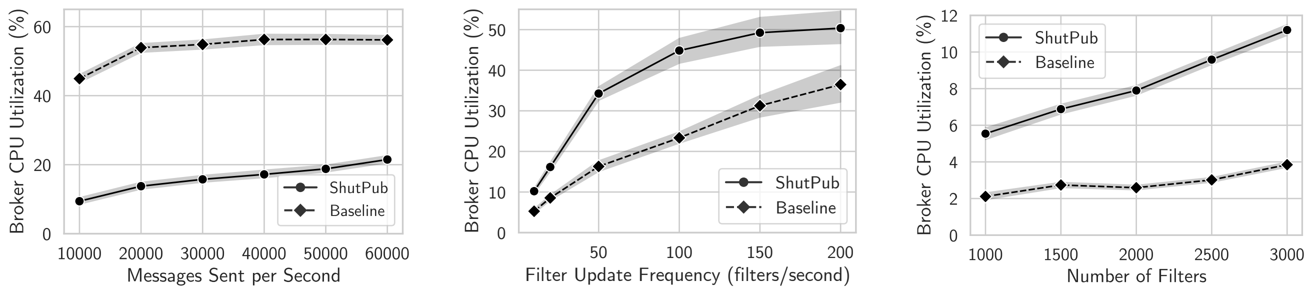
The frequency with which publishers send messages affects the broker load. As ShutPub filters messages on the publisher side, increased frequency should reduce the load on the broker compared to the baseline. To evaluate this, we perform experiments with 100 publishers and 4 subscribers connected to the broker, and vary the total publisher message sending frequency from 10,000 to 60,000 messages per second in steps of 10,000. The four subscribers subscribe to subtopics of the same topic. Each publisher publishes 20,000 messages to the same topic with on average every tenth message matching a subscription. We show the results in Figure 4a.

As the message frequency increases, the CPU utilization increases for both ShutPub and baseline. ShutPub average CPU utilization is 37.67 pp lower than the baseline, which is a reasonable improvement.

5.4 Broker Overheads: Updating Filters

In this experiment, we study the effect of frequent filter updates resulting from subscription changes. For this, we use 10 subscribers with a unique topic for each and vary their respective filter update frequency (10, 20, 50, 100, 150, and 200 times per second) across experiment runs. We show the resulting CPU utilization in Figure 4b.

As the filter frequency increases, the CPU utilization for both ShutPub and baseline increases. ShutPub has a higher CPU utilization since it needs to maintain the filters. Updating filters 200 times per second yields an average CPU



(a) Message Frequency **(b) Filter Update Frequency** **(c) Number of Filters**

Figure 4: The load on the broker increases for both baseline and ShutPub in all three experiments. For message processing, ShutPub significantly reduces the CPU load compared to the baseline (4a). Filter updates (Figure 4b) from the subscribers and the total number of filters maintained result in a small CPU overhead compared to the baseline (Figure 4c).

utilization overhead of 12.33 pp compared to the baseline scenario. Considering that this is already a rather extreme corner case, we consider this overhead acceptable for most scenarios – especially in comparison to the benefits.

5.5 Broker Overheads: Creating Filters

As more subscribers connect, the broker needs to maintain more filters, which increases the load on the broker. We perform experiments with 500, 1,000, 1,500, 2,000, 2,500 and 3,000 subscribers, where each subscriber has a unique filter. We then monitor the CPU utilization of the broker and compare it to the baseline using ActiveMQ.

The results in Figure 4c show that as more subscribers connect, both the baseline and ShutPub broker CPU utilization increase. On average, ShutPub has an average 5.36 pp higher load. However, even when 3,000 subscribers connect to the ShutPub broker running on a rather small `e2-standard-2` (2 vCPUs, 8GB RAM), the CPU utilization is 11.2% which is still at a low level. For realistic edge computing scenarios, the overhead of creating filters in ShutPub will not be relevant.

5.6 Discussion

To showcase the performance of ShutPub, we conducted experiments in terms of network load, publisher and broker overheads. Since changes in either publishers or subscribers affect the broker overhead, we extensively investigate the broker overhead by varying the number of filters, filter update frequency, and message throughput from publishers.

Lower network load between publishers and broker. ShutPub allows publishers to only distribute messages when there is a need from subscribers with a negligible communication overhead. This approach improves network efficiency, ensures stable message transmission in network-constrained situations, and reduces cost in the edge environment.

Reduced resource consumption on publishers. The average CPU utilization of ShutPub publishers decreases significantly as the number of messages sent decreases, even though they need to perform additional message filtering. When the CPU utilization of the baseline publisher reaches 89.35%, the CPU utilization of ShutPub is only 36.93%. Consequently, ShutPub enables publishers to (i) support more intensive message-sending scenarios, (ii) operate in a more energy-critical environment, and (iii) save energy consumption, thus having a longer lifetime if they are battery-powered.

Increased broker performance. ShutPub broker shows better performance on publisher message throughput. However, as ShutPub broker needs to further maintain filters for publishers, there is additional broker overhead compared to the vanilla ActiveMQ broker. On a resource-limited `e2-standard-2` broker, the extreme corner case overhead is around ten percent. For realistic scenarios, brokers have better capabilities which means that the overhead from filters is negligible and the throughput advantage remains. Consequently, ShutPub allows the broker to have a higher performance by either having more publisher connections or having less broker resource utilization.

6 Conclusion

Content-based pub/sub systems allow subscribers to precisely express which messages they are interested in. In practice, publishers send all their messages to a broker where messages without subscribers are discarded. This wastes a lot of resources which can be crucial, especially when moving towards the edge.

In this paper, we proposed ShutPub, a publisher-side middleware that automatically shifts message filtering logic from the broker to the publishers in order to drop unnecessary messages already at their source. We implement a proof-of-concept prototype based on ActiveMQ and show through experiments that ShutPub can lead to significant resource savings depending on the distribution of interests. The results show that ShutPub generally improves the performance of network bandwidth on the publisher side, as well as reducing CPU utilization for both publishers and the broker.

Acknowledgments

Funded by the Bundesministerium für Digitales und Verkehr (BMDV, German Federal Ministry for Digital and Transport) – 19F1119A.

References

[1] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. 2001. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.* 19, 3 (Aug. 2001), 332–383. doi:10.1145/380749.380767

[2] Lisi Chen, Shuo Shang, Zhiwei Zhang, Xin Cao, Christian S. Jensen, and Panos Kalnis. 2018. Location-Aware Top-k Term Publish/Subscribe. In *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering* (Paris, France) *(ICDE '18)*. IEEE, New York, NY, USA, 749–760. doi:10.1109/ICDE.2018.00073

[3] Kyu Bong Cho, Sung Keun Song, and Hee Yong Youn. 2007. Publisher-side Event Filtering for QoS-Awareness in Ubiquitous Computing. In *Proceedings of the 2007 International Conference on Computational Science and its Applications* (Kuala Lumpur, Malaysia) *(ICCSA '07)*. IEEE, New York, NY, USA, 361–366. doi:10.1109/ICCSA.2007.51

[4] Ivan Čilić and Ivana Podnar Žarko. 2022. Adaptive Data-Driven Routing for Edge-to-Cloud Continuum: A Content-Based Publish/Subscribe Approach. In *Global IoT Summit*. Springer, 29–42.

[5] Cédric du Mouza and Nicolas Travers. 2018. Relevant Filtering in a Distributed Content-based Publish/Subscribe System. In *NoSQL Data Models: Trends and Challenges*. Wiley Data and Cybersecurity, 203–244.

[6] Eli Fidler, Hans-Arno Jacobsen, Guoli Li, and Serge Mankovski. 2005. The PADRES Distributed Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*. IGI Global, 12–30.

[7] Daniel Happ and Suzan Bayhan. 2020. On the impact of clustering for IoT analytics and message broker placement across cloud and edge. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking* (Heraklion, Greece) *(EdgeSys '20)*. Association for Computing Machinery, New York, NY, USA, 43–48. doi:10.1145/3378679.3394538

[8] Jonathan Hasenburg and David Bermbach. 2020. DisGB: Using Geo-Context Information for Efficient Routing in Geo-Distributed Pub/Sub Systems. In *Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing* (Leicester, United Kingdom) *(UCC 2020)*. IEEE, New York, NY, USA, Dec. doi:10.1109/UCC48980.2020.00026

[9] Jonathan Hasenburg and David Bermbach. 2020. GeoBroker: Leveraging Geo-Context for IoT Data Distribution. *Elsevier Computer Communications* 151 (Feb. 2020), 473–484. doi:10.1016/j.comcom.2020.01.015

[10] Jonathan Hasenburg, Florian Stanek, Florian Tschorsch, and David Bermbach. 2020. Managing Latency and Excess Data Dissemination in Fog-Based Publish/Subscribe Systems. In *Proceedings of the Second IEEE International Conference on Fog Computing* (Sydney, NSW, Australia) *(ICFC 2020)*. IEEE, New York, NY, USA, 9–16. doi:10.1109/ICFC49376.2020.00010

[11] Sayed Hadi Hashemi, Faraz Faghri, Paul Rausch, and Roy H Campbell. 2016. World of empowered IoT users. In *Proceedings of the 2016 IEEE First International Conference on Internet-of-Things Design and Implementation* (Berlin, Germany) *(IoTDI '16)*. IEEE, New York, NY, USA, 13–24. doi:10.1109/IoTDI.2015.39

[12] Yafei Li, Lei Gao, Haobo Sun, Huiling Li, and Qingshun Wu. 2022. PRID: An Efficient Pub/Sub Ride Hitching System. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (Atlanta, GA, USA) *(CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 4921–4925. doi:10.1145/3511808.3557213

[13] Yanhong Li, Wang Zhang, Rongbo Zhu, Guohui Li, Maode Ma, Lihchyun Shu, and Changyin Luo. 2019. Fog-Based Pub/Sub Index With Boolean Expressions in the Internet of Industrial Vehicles. *IEEE Transactions on Industrial Informatics* 15, 3 (Sept. 2019), 1629–1642. doi:10.1109/TII.2018.2868720

[14] Pezhman Nasirifard and Hans-Arno Jacobsen. 2022. A Serverless Publish/Subscribe System. (Oct. 2022). arXiv:2210.07897

[15] Shunya Nishio, Daichi Amagata, and Takahiro Hara. 2022. Lamps: Location-Aware Moving Top-k Pub/Sub. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (March 2022), 352–364. doi:10.1109/TKDE.2020.2979176

[16] Shiyou Qian, Jian Cao, Yanmin Zhu, and Minglu Li. 2014. Rein: A fast event matching approach for content-based publish/subscribe systems. In *Proceedings of the IEEE Conference on Computer Communications* (Toronto, ON, Canada) *(INFOCOM 2014)*. IEEE, New York, NY, USA, 2058–2066. doi:10.1109/INFOCOM.2014.6848147

[17] Shiyou Qian, Jian Cao, Yanmin Zhu, Minglu Li, and Jie Wang. 2014. H-tree: An efficient index structure for event matching in content-based publish/subscribe systems. *IEEE Transactions on Parallel and Distributed Systems* 26, 6 (May 2014), 1622–1632. doi:10.1109/TPDS.2014.2323262

[18] Shiyou Qian, Weichao Mao, Jian Cao, Frédéric Le Mouël, and Minglu Li. 2019. Adjusting Matching Algorithm to Adapt to Workload Fluctuations in Content-based Publish/Subscribe Systems. In *Proceedings of the IEEE Conference on Computer Communications* (Paris, France) *(INFOCOM 2019)*. IEEE, New York, NY, USA, 1936–1944. doi:10.1109/INFOCOM.2019.8737647

[19] Thomas Rausch, Stefan Nastic, and Schahram Dustdar. 2018. EMMA: Distributed QoS-Aware MQTT Middleware for Edge Computing Applications. In *Proceedings of the 2018 IEEE International Conference on Cloud Engineering* (Orlando, FL, USA) *(IC2E)*. IEEE, New York, NY, USA, 191–197. doi:10.1109/IC2E.2018.00043

[20] Wanghua Shi and Shiyou Qian. 2022. HEM: A Hardware-Aware Event Matching Algorithm for Content-Based Pub/Sub Systems. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA 2022)*. Springer, Cham, Switzerland, 277–292. doi:10.1007/978-3-031-00123-9_23

[21] Minghe Wang, Trever Schirmer, Tobias Pfandzelter, and David Bermbach. 2023. Lotus: Serverless In-Transit Data Processing for Edge-based Pub/Sub. In *Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking* (Rome, Italy) *(EdgeSys '23)*. ACM, New York, NY, USA. doi:10.1145/3578354.3592869

[22] Kaiwen Zhang, Mohammad Sadoghi, Vinod Muthusamy, and Hans-Arno Jacobsen. 2017. Efficient covering for top-k filtering in content-based publish/subscribe systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (Las Vegas, NV, USA) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 174–184. doi:10.1145/3135974.3135976