A Study of Publish/Subscribe Middleware Under Different IoT Traffic Conditions

Zhuangwei Kang
Vanderbilt University
Nashville, Tennessee
zhuangwei.kang@vanderbilt.edu

Robert Canady
Vanderbilt University
Nashville, Tennessee
robert.e.canady@vanderbilt.edu

Abhishek Dubey
Vanderbilt University
Nashville, Tennessee
abhishek.dubey@vanderbilt.edu

Aniruddha Gokhale
Vanderbilt University
Nashville, Tennessee
a.gokhale@vanderbilt.edu

Shashank Shekhar
Siemens Technology
Princeton, New Jersey
shashankshekhar@siemens.com

Matous Sedlacek
Siemens Technology
Munich, Germany
matous.sedlacek@siemens.com

Abstract

Publish/Subscribe (pub/sub) semantics are critical for IoT applications due to their loosely coupled nature. Although OMG DDS, MQTT, and ZeroMQ are mature pub/sub solutions used for IoT, prior studies show that their performance varies significantly under different load conditions and QoS configurations, which makes middleware selection and configuration decisions hard. Moreover, the load conditions and role of QoS settings in prior comparison studies are not comprehensive and well-documented. To address these limitations, we (1) propose a set of performance-related properties for pub/sub middleware and investigate their support in DDS, MQTT, and ZeroMQ; (2) perform systematic experiments under three representative, lab-based real-world IoT use cases; and (3) improve DDS performance by applying three of our proposed QoS properties. Empirical results show that DDS has the most thorough QoS support, and more reliable performance in most scenarios. In addition, its Multicast, TurboMode, and AutoThrottle QoS policies can effectively improve DDS performance in terms of throughput and latency.

CCS Concepts: • **Software and its engineering** → **Message oriented middleware**; **Publish-subscribe / event-based architectures**; • **General and reference** → *Evaluation*.

M4IoT'20, December 7–11, 2020, Delft, Netherlands

ACM ISBN 978-1-4503-8205-2/20/12…$15.00
https://doi.org/10.1145/3429881.3430109

Keywords: Publish/Subscribe Middleware, Benchmarking, MQTT, DDS, ZeroMQ, Performance Evaluation

1 Introduction

Distributed deployment of real-time applications and high-speed dissemination of massive data have been hallmarks of the Internet of Things (IoT) platforms. IoT applications typically adopt publish/subscribe (pub/-sub) middleware for asynchronous and cross-platform communication. OMG Data Distribution Service (DDS), ZeroMQ, and MQTT are three representative pub/sub technologies that have entirely different architectures (decentralized data-centric, decentralized message-centric, and centralized message-centric, respectively). All of them implement the pub/sub messaging pattern and provide a set of configurable parameters for customizing middleware behaviors and resource allocation. Accordingly, an essential question that needs to be answered is how to choose an appropriate middleware given a workload condition, and which parameters should be regarded for making the optimal configuration decisions.

Therefore, this study makes three major contributions to research on this problem:

1. We propose a set of QoS properties that are tied to the performance of IoT pub/sub applications and investigate which of those are supported by DDS, MQTT, and ZeroMQ.
2. We then conduct a systematic set of experiments to assess their performance in three pub/sub use cases (high-frequency, periodic, and sporadic), which provide us with baselines for doing further performance optimization.
3. Since empirical results show that DDS has the most stable performance in the above scenarios and provides the most in-line support to the QoS properties we proposed, we further experimentally validate the impact of three competitive QoS policies (Multicast, TurboMode, and AutoThrottle)

on improving DDS application performance under such conditions.

2 Essential QoS Properties for IoT Systems

The primary purpose of this section is to introduce several performance-related QoS policies available in DDS, MQTT, and ZeroMQ that are necessary for performance-sensitive IoT applications. Although industrial middleware vendors provide considerable adjustable QoS properties for satisfying diverse demands in efficiency, reliability, security, durability, etc., only a small portion of them have an apparent influence on application performance (throughput, latency) in normal use cases. This section empirically selects those that can effectively change inbound/outbound network traffic or message processing logic due to the fact that the performance of pub/sub middleware depends mainly on the network congestion status and internal resource utilization.

2.1 Reliability

Reliability policy determines whether messages can be reliably transmitted, which fatally affect the data-processing and decision-making processes of IoT applications. DDS provides two types of transmission guarantees: Best Effort (i.e., no guarantees) and Reliable (i.e., retry until success). MQTT provides three service levels: QoS 0, QoS 1, QoS 2, representing "at-most-once", "at-least-once", and "exactly-once", respectively. MQTT's QoS 0 is equivalent to DDS' BestEffort and QoS 2 to DDS' Reliable delivery. ZMQ pub/sub pattern may lose messages when the network is congested.

2.2 Multicast

Multicast allows IoT applications to scale better when multiple subscribers exist. Both DDS and ZeroMQ allow users to set the number of hops that multicasting messages can traverse, which intends to avoid flooding large networks with multicast traffic. Although MQTT runs over unicast protocol, it emulates the application-layer multicast by wrapping up point-to-point TCP connections, which is also known as multicast-over-unicast. The protocol utilized in ZeroMQ for multicast include Pragmatic General Multicast (PGM) [5] and Encapsulated Pragmatic General Multicast (EPGM).

2.3 Intelligent Batching

Message batching intends to improve the throughput of IoT applications. It avoids frequent system calls through the network stack due to message by message processing and can be performed at a fixed interval or over number of messages. The TurboMode QoS in DDS automatically decides the optimal number of bytes in a batch on publisher side based on the data sample size, writing speed and the real-time system state. Message batching in ZeroMQ can be enabled at either publisher or subscriber side. Rather than deciding timeout or batch size for a single chunk, ZeroMQ always forwards all messages queued in memory at the moment to the network interface card in one go. However, due to the lack of strict reliability guarantees in ZeroMQ's pub/sub pattern, messages sent in a large batch risk being lost when the network is crowded. At the subscriber side, if the subscriber can keep up with publisher and there is no application-level queue backlog, ZeroMQ will turn off its batching function for the sake of lower end-to-end delay. Otherwise, the batching will be enabled to help the worker thread promote the speed of flushing backlogged messages. MQTT does not support intelligent batching.

2.4 Rate Limit

Rate Limit is a flow control mechanism that protects network resources from being encroached by malicious actors in an IoT cluster by specifying the maximum rate at which a publisher may send samples to the network. DDS allows users to define the Custom Flow Controller that maintains a separate FIFO queue for each connection in which data instances generated by asynchronous publishers can be placed. ZeroMQ supports rate limit only when the multicast protocol (PGM/EPGM) is enabled, and is implemented by setting the multicast window size. MQTT server shapes the egress traffic of each TCP/SSL connection based on the Leaky Bucket (LB) algorithm.

2.5 Message LifeSpan

Message expiration mechanism restrains the longest time that a message is regarded as a legal one in the system, which improves overall memory utilization rate and avoids delivering stale data. In the reliable transport mode of DDS, if the LifeSpan QoS policy is not configured, unconfirmed data instances will reside in the publisher's writing buffer for a long time and be re-transmitted endlessly, which not only wastes memory and bandwidth, but also destroys the timeliness of messages. Adhering to similar intent, MQTT implementations (EMQ X, HiveMQ, and VerneMQ) that adopt the latest MQTT specification (MQTT 5.0) allow publishers to specify a property for each sample called message-expiry-interval that defines the actual lifespan of messages in seconds. There is no analog to LifeSpan QoS in current ZeroMQ implementations.

2.6 Topic Priority

Topic priority aims to distinguish the importance of different messages, which ensures that the overall utility of a multi-topic IoT system is maximized under given resource limitation. DDS enables this property via the Tansport_Priority policy that is signified by a 32-bit signed integer. An analogous parameter in ZeroMQ is called ZMQ_TOS that is implemented in the same logic

as DDS. Unlike the above system-level implementations, the topic priority in MQTT applications is designated as a configurable attribute of the message queue in the broker with values from 0 to 255.

3 Understanding the Performance Baseline

The purpose of this section is to understand the baseline performance of DDS, MQTT, and ZeroMQ under three representative IoT workload conditions (high-frequency, periodic, sporadic). We performed our tests on an ARM cluster comprising 10 Raspberry Pi 3 Model B boards that have 1.20 GHz CPU speed, four physical cores, and 1 GB memory. Software details of the cluster are as follows: Raspbian 9 OS, Linux 4.14.91 kernel, and GCC 4.7.3. The cluster bandwidth of 95 Mbps was inferred using Linux iperf3. Interference was minimized by pinning publishers and subscribers to separate cores. To avoid latency measurement error introduced by system clock jitters, we synchronized the system clock on each node using the Precision Time Protocol (PTP) [4]. Our test results indicate that PTP can guarantee the clock offset within 200 microseconds between Raspberry Pi boards even if the CPU is 80 percent utilized.

We leveraged RTI-Perftest [7] to monitor and benchmark throughput, latency and CPU utilization of the testing DDS application. RTI-Perftest is a highly configurable command-line benchmarking tool developed by RTI for evaluating performance of applications that use RTI Connext DDS Professional 6.0[1] as middleware. RTI-Perftest measures throughput by counting the amount of bytes received by the subscriber per second. To avoid measurement errors caused by system clock jitter, RTI-Perftest calculates the one-way delay by sending latency test data samples in a ping-pong manner. For MQTT and ZeroMQ, we developed custom testing tools using open source APIs. All tests were repeated five times and each run lasted 90 seconds. We measured performance metrics every 5 seconds.

3.1 High-frequency Data-flow Tests

This test was motivated by a fault diagnosis system where vibrations and sound information gathered by acoustic sensors are continuously propagated to ADC or cloud servers for further ML-based analysis. Hence, in this test, a publisher floods a continuous data-flow to single/multiple subscribers with unlimited rate, which were profiled with default QoS settings. Figure 1 plots mean throughput versus payload size.

The 1-1 subplot indicates that application throughput grows as the payload size increases. When the message is smaller than 1 KB, ZeroMQ performs best, whose throughput is 17.15x-30.69x higher than that of MQTT and 18.8%-110.66% greater than DDS. The reason is

[1] https://www.rti.com/products/connext-6

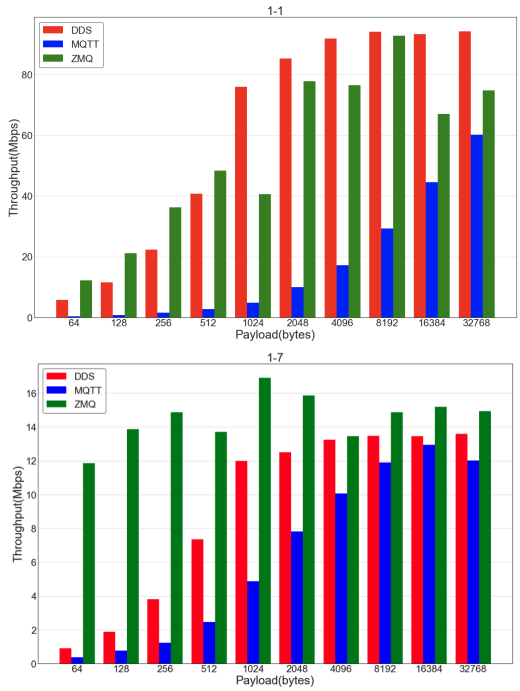

Figure 1. High-frequency Data-flow Tests: 1pub-1sub & 1pub-7sub, PubRate: unlimited. Figure shows mean throughput over five runs. QoS settings: Unicast, Reliable, No Batching.

there is no bandwidth or CPU pressure for either ZeroMQ or DDS at this time, and since ZeroMQ is built on top of the socket layer (lower than DDS), it spends less time than DDS in executing application-level data processing. However, when the message is larger than 1KB, ZeroMQ throughput is not smoothly converged, and its throughput becomes lower than DDS. The shape of the DDS curve is as expected: its throughput increases first then converges to the maximum bandwidth (95mbps). Compared with DDS and ZeroMQ, the throughput of MQTT is poor due to its broker-centric architecture. In the 1-7 test, the throughput of DDS and MQTT is one-seventh of the physical bandwidth since messages are forwarded to seven subscribers one by one. On the other hand, ZeroMQ gains the highest throughput, which reveals that the overhead of ZeroMQ packets in one-to-many cases is significantly lower than DDS and MQTT.

3.2 Periodic Data-flow Tests

The Periodic use case maps to the continuous data-flow in real-life, where the size and frequency of data change less frequently, e.g., a wind farm monitoring system in which windmills that rotate at a constant speed send telemetry data at a fixed rate through which vendors can continuously track the operational status of windmills. In this kind of scenario, latency is more important than throughput as the timeliness of monitoring data is more

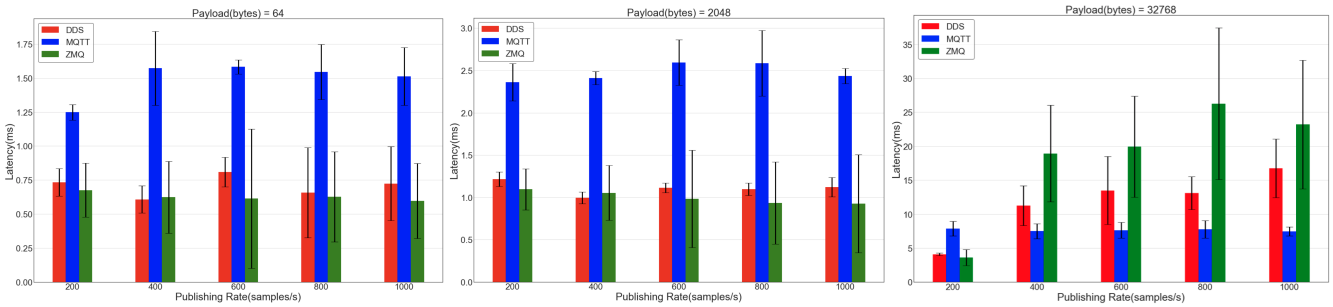

Figure 2. Periodic Data-flow Tests: 1pub-1sub. Figure shows 90th latency over five runs. QoS settings: Unicast, Reliable, No Batching.

important. In the following test, we simulated this use case and configured a publisher to send messages to a subscriber at a specified rate. The publishing rate varies from 200 to 1000 samples per second. Figure 2 shows the results of the 90th percentile latency versus publishing rate for small, medium, and large messages.

For small (64B) and medium (2KB) messages, MQTT latency slightly improves first then remains flat, the reason being that MQTT broker needs more time to process ingress/egress traffic as the sending rate increases. However, when the broker can not keep up with the publisher and the writing buffer on the publisher node is exhausted, the writing process of publisher will be blocked, thereby keeping the actual message dissemination rate and latency unchanged. In addition, the change of latency is not very obvious for DDS and ZeroMQ because (1) their participants connect in an end-to-end manner, (2) subscriber can keep up with the publisher since bandwidth is not stressed.

For large samples (32KB), MQTT latency remains flat as the publishing rate varies, and is consistently at the lowest level compared to others when the dissemination rate is faster than 400 samples per second for similar reasons as before. DDS and ZeroMQ share the same trend with DDS latency a bit larger than that of ZeroMQ when the publishing rate is 200 samples/sec. ZeroMQ and DDS latency suddenly increase as the message diffusion rate pass 400 and 600 samples per second, respectively, and then fluctuates marginally. We believe the reasons are twofold: (1) the bandwidth is exhausted when the publishing rate surpasses 400 samples per second (400 samples/s * 32KB > 95Mbps); (2) the publishing process is blocked as the local buffer is exhausted, which is the same reason as in MQTT.

3.3 Sporadic Data-flow Tests

Multiple independent pub/sub applications may co-locate in the same LAN environment. Consider the same wind farm monitoring system example, if anomalous status is identified on some windmills (i.e., stops spinning due to slow wind speeds or mechanical stoppage), it is necessary to send a large volume of failure detection data to the central control system. As a result, co-located applications may experience latency deterioration due to bandwidth contention caused by the bursty traffic.

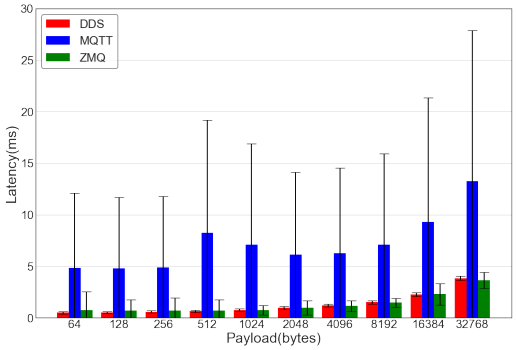

Figure 3. Sporadic Data-flow Test: 1pub-1sub. Figure shows the 90th latency of the PDF application, standard deviation on each bar indicates the interference caused by the co-located SDF application. Payload(SDF):2MB, PubRate(SDF):unlimited, PubRate(PDF):25Mbps

To that end, we generated a periodic data-flow (PDF) and a sporadic data-flow (SDF) using two one-to-one pub/sub applications with separate topics. The packets size of SDF application was set to 2MB and the message frequency of PDF set to 25Mbps of the physical bandwidth. Also, the SDF application began 30 seconds later than the PDF application and executed for 10 seconds.

Figure 3 depicts the latency of the application under the interference of the SDF application. MQTT latency has higher standard deviation than DDS and MQTT, which implies MQTT is more sensitive to the bursty data stream due to the presence of the broker. Moreover, DDS tolerates the situation better than ZeroMQ.

4 DDS-focused Evaluations

Since DDS provides more modularized and pluggable QoS properties, this section probes it further. Prior

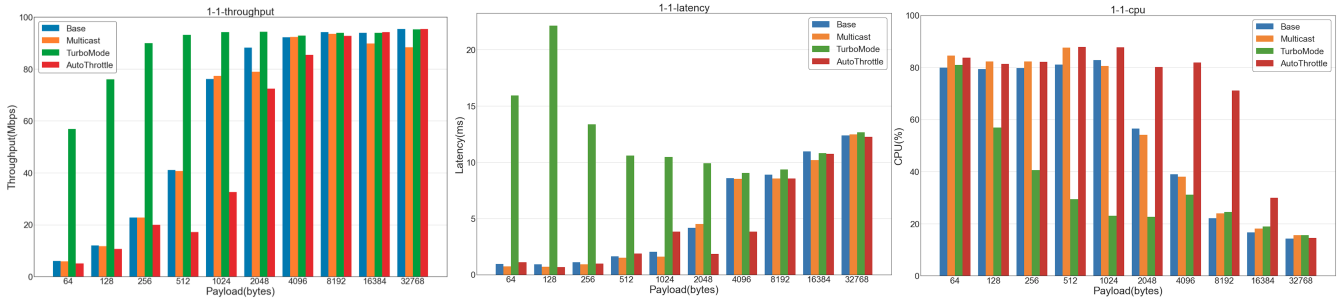

Figure 4. DDS QoS Test: 1pub-1sub, PubRate: unlimited. Figure shows mean throughput, latency and CPU utilization. QoS(Base): Unicast, Reliable, No Batching; QoS(Multicast): Multicast, Reliable, No Batching; QoS(TurboMode): Unicast, Reliable, TurboMode; QoS(AutoThrottle): Unicast, Reliable, No Batching, AutoThrottle.

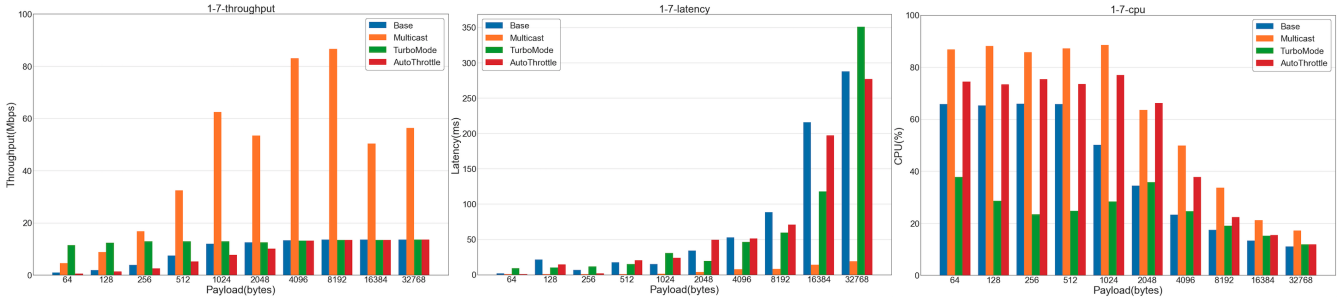

Figure 5. DDS QoS Test: 1pub-7sub, PubRate: unlimited. Figure shows mean throughput, latency and CPU utilization. QoS(Base): Unicast, Reliable, No Batching; QoS(Multicast): Multicast, Reliable, No Batching; QoS(TurboMode): Unicast, Reliable, TurboMode; QoS(AutoThrottle): Unicast, Reliable, No Batching, AutoThrottle.

works [2][6] reveal that transport protocol, message batching, and flow control are non-trivial aspects of performance tuning in networked applications, we evaluate DDS along these dimensions that corresponds to the Multicast, TurboMode, and AutoThrottle QoS policies in DDS. The setup was similar as in the High-frequency Data-flow Test.

Figures 4 and 5 reveal that the publisher's CPU usage remains flat in the beginning then gradually decreases as the message size increases, which indicates that messages are produced less often as message size increases since data samples need more time to be delivered. Yet, the writing process is blocked during this period until more packets can be put into the pipe.

The TurboMode improves throughput 23.6%-848.3% and 7.5%-1166.7% for messages that are smaller than 1KB in 1-1 and 1-7 tests, respectively, as the network stack is traversed less frequently. Moreover, it does not always lead to higher latency in the 1-n test because the time consumed on sending individual messages to multiple receivers in a unicast manner may be longer than sending multiple messages in batch. Multicast effectively improves application performance in the 1-7 test (272.3%-842.2%) but the throughput cannot converge to the physical bandwidth as payload increases, which we believe is due to the limitation of the network switch.

When the AutoThrottle mode is enabled, our results show that the application throughput reduced an average of 18.0% and 18.7% in the 1-1 and 1-7 tests, respectively. Likewise, the latency reduced 3.6% and 8.8%, respectively. Since the AutoThrottle feature needs to keep tracking the system states (send window occupancy and NACK messages amount) to make throttling decisions, it results in higher CPU utilization than the normal configuration.

5 Related Work

Performance assessment of pub/sub middleware have been conducted by many prior works under different experimental settings. Pereira et al. [9] propose a set of qualitative and quantitative dimensions for benchmarking IoT middleware. They used a large dataset to simulate a smart city use case, and evaluated the performance of two middleware platforms (FIWARE[2] and oneM2M[3]) from their proposed benchmarking dimensions. Similarly, the quantitative analysis in [1] presents performance (throughput and latency) variation between

[2]https://www.fiware.org.
[3]http://www.onem2m.org.

Open MQ, Active MQ, and Mantaray MQ for different message sizes. However, since the middleware they investigated are all broker-based, they learned publishing and subscribing processes separately, rather than performing end-to-end tests.

In [10], authors provided an overview of round trip time (RTT) difference of OPC UA, ROS, DDS, and MQTT under different CPU and network load conditions. Dobbelaere et al.[3] established a generic comparison framework based on the core functionality of pub/sub systems. Using this framework, they delved into qualitative and quantitative comparison of two commercially-supported middleware: Kafka and RabbitMQ. To avoid interference from the network layer, they executed their experiments on a single host with empirical application configurations. Luzuriaga et al.[8] present an experimental evaluation of AMQP and MQTT in the context of unstable network conditions. Their assessments are based on a simple one-to-one publish/subscribe scenario. Compared to these works, we designed test cases based on real-life scenarios and provided users with more insightful guidelines on selecting QoS policies to improve middleware performance.

6 Concluding Remarks

This paper empirically evaluates the performance of three pub/sub technologies: OMG DDS, MQTT and ZeroMQ for representative IoT scenarios (high-frequency, periodic, and sporadic). DDS provides more comprehensive and modularized QoS support than others, and also demonstrates better overall latency and throughput in most evaluated scenarios. Specifically, DDS gained higher throughput than ZeroMQ and MQTT in the high-frequency data-flow use case. In periodic data-flow, ZeroMQ has lower latency than DDS for small (64B) and medium (2KB) messages. DDS latency outperforms ZeroMQ when sending large messages (32KB). MQTT is more sensitive to the in-parallel sporadic data-flow, and DDS can successfully shield the interference. Our results also reveal that DDS' Multicast QoS can effectively improve throughput in multi-subscriber scenarios. The TurboMode property can intelligently decide the appropriate batch size with regard to different payload and significantly improve throughput for small messages. Moreover, the AutoThrottle property results in lower throughput and latency and higher CPU utilization.

Our future work will include: (1) examining more QoS settings (not only for DDS, but also MQTT and ZeroMQ) experimentally; (2) evaluating more middleware using real-world workloads instead of synthetic data-flows; and (3) designing intelligent decision-making algorithms to automatically configure and adaptively adjust middleware QoS parameters under various (dynamic) load conditions.

Acknowledgments

The authors thank all insightful comments from the anonymous reviewers of the M4IoT'20. We also appreciate valuable suggestions from our colleagues at Vanderbilt University, including Dr.Shweta Khare and Scott Eisele. This work is supported in part by funding from Siemens Technology and Cisco. Any opinions, findings, and conclusions or recommendations expressed in this material are of the author(s) and do not necessarily reflect the views of the sponsors.

References

[1] Sanjay P Ahuja and Naveen Mupparaju. 2014. Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware. *Computer and information science* 7, 4 (2014), 9.

[2] Sowmya Balasubramanian, Dipak Ghosal, Kamala Narayanan Balasubramanian Sharath, Eric Pouyoul, Alex Sim, Kesheng Wu, and Brian Tierney. 2018. Auto-tuned publisher in a pub/sub system: Design and performance evaluation. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 21–30.

[3] Philippe Dobbelaere and Kyumars Sheykh Esmaili. 2017. Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. In *Proceedings of the 11th ACM international conference on distributed and event-based systems*. 227–238.

[4] John C Eidson, Mike Fischer, and Joe White. 2002. IEEE-1588™ Standard for a precision clock synchronization protocol for networked measurement and control systems. In *Proceedings of the 34th Annual Precise Time and Time Interval Systems and Applications Meeting*. 243–254.

[5] Jim Gemmell, Todd Montgomery, Tony Speakman, and Jon Crowcroft. 2003. The PGM reliable multicast protocol. *IEEE network* 17, 1 (2003), 16–22.

[6] Joe Hoffert, Aniruddha Gokhale, and Douglas C Schmidt. 2011. Timely autonomic adaptation of publish/subscribe middleware in dynamic environments. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)* 2, 4 (2011), 1–24.

[7] Real-Time Innovations. 2020. rtiperftest. https://github.com/rticommunity/rtiperftest.

[8] Jorge E Luzuriaga, Miguel Perez, Pablo Boronat, Juan Carlos Cano, Carlos Calafate, and Pietro Manzoni. 2015. A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, 931–936.

[9] Carlos Pereira, João Cardoso, Ana Aguiar, and Ricardo Morla. 2018. Benchmarking Pub/Sub IoT middleware platforms for smart services. *Journal of Reliable Intelligent Environments* 4, 1 (2018), 25–37.

[10] Stefan Profanter, Ayhun Tekat, Kirill Dorofeev, Markus Rickert, and Alois Knoll. 2019. OPC UA versus ROS, DDS, and MQTT: performance evaluation of industry 4.0 protocols. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*.