PSMark: a distributed IoT benchmark for publish/subscribe under domain-based workloads

Christian Badolato, Nathan Samson, Houssam Hajj Hassan, Chih-Kai Huang, Georgios Bouloukakis, Primal Pappachan, Roberto Yus

PSMark: A Distributed IoT Benchmark for Publish/Subscribe Under Domain-Based Workloads

Christian Badolato[†], Nathan Samson[†], Houssam Hajj Hassan[**‡], Chih-Kai Huang[¶‡],
Georgios Bouloukakis[*], Primal Pappachan[§], Roberto Yus[†]
{cbad1, nsamson4, ryus}@umbc.edu, houssam.hajjhassan@orange.com, chih-kai.huang@telecom-paris.fr
gbouloukakis@upatras.gr, primal@pdx.edu
[†]University of Maryland, Baltimore County, USA; [‡]Télécom SudParis, Institut Polytechnique de Paris, France;
[¶]Télécom Paris; [*]University of Patras, Greece; [§]Portland State University, USA; [**]Orange Innovation, France

***Abstract*—The Publish/Subscribe (pub/sub) paradigm is widely used in the Internet of Things (IoT). Standalone sensors, wearables, and other devices act as producers that publish messages to consumers such as edge servers or even other IoT devices. Selecting and configuring a pub/sub protocol for an IoT system requires considering network requirements, device reliability, and required Quality-of-Service guarantees. Pub/sub benchmarking suites can help compare expected behavior of various protocols, implementations, and network configurations. However, current pub/sub benchmarks focus primarily on stress testing systems assuming mostly static configurations of homogeneous publishers which are not representative of real-world IoT deployments. To address this, we present PSMark, a distributed, multi-protocol benchmark for evaluating topic-filtered pub/sub systems under workloads representative of real-world IoT environments. PSMark supports (i) workloads representative of heterogeneous IoT device deployments including variations in device communication parameters, (ii) evaluation of distributed IoT deployments with multiple data aggregation servers, (iii) cross-protocol measurements across MQTT and DDS, with extensibility to additional protocols, and (iv) a modular design for adding additional metrics and interfaces. We further construct twelve IoT-focused workloads derived from seven real-world datasets in the domains of manufacturing, healthcare, smart homes, and smart cities. Finally, we benchmark five popular MQTT brokers and one DDS implementation using PSMark and analyze their performance across multiple testbeds and Quality-of-Service settings.**

***Index Terms*—Internet of Things, Benchmarking, Publish/Subscribe, MQTT, DDS, Performance evaluation**

I. Introduction

The Internet of Things (IoT) is now integral to modern data processing, with an estimated 21.5 billion devices worldwide in 2025 [1]. These devices span smart cities, homes, healthcare, agriculture, and the Industrial Internet of Things (IIoT) [2]. Adoption is expected to continue to grow as the Artificial Intelligence of Things (AIoT) enables advances in transportation, factory maintenance, medical diagnoses, and more [3], [4], [5]. Given the volume of data produced, transmitted, and consumed within IoT networks, efficient data exchange is critical. Publish/Subscribe (pub/sub) protocols are widely recommended for scalable, lightweight, decoupled communication in distributed IoT systems. Both centralized, broker-based protocols (e.g., MQTT [6]) and decentralized, brokerless protocols (e.g., the Data Distribution Service–DDS [7]) have seen significant adoption in IIoT [8], [9], healthcare [10], and smart cities [10], [11], [12]. However, pub/sub protocols and their implementations are not one-size-fits-all: selecting a pub/sub system and tuning its communication parameters must align the system's features with domain and use case requirements (e.g., timing and reliability).

This is a particularly important consideration during the design and deployment of pervasive IoT systems. Consider a traffic management and safety deployment in which a large number of traffic signals must use a pub/sub system to ensure fault-tolerant, real-time monitoring and control of traffic patterns [12]. In this environment, the pub/sub system must be able to efficiently handle a vast number of clients without experiencing timing delays due to connection management or the exhaustion of computational resources. Likewise, a deployment for monitoring hospital patients requires constant, reliable data transfer between medical devices, processing servers, and status displays to ensure prompt and accurate response to patient emergencies [13]. Despite the shared requirements of low-latency and reliable messaging, these use cases likely involve vastly different device types, device quantities, data volumes, and processing power. IoT system designers must thoroughly evaluate various pub/sub systems against each of these factors before selecting one for deployment.

Benchmarking frameworks have been established as an important mechanism to evaluate the performance of modern technologies and smart spaces [14], [15], [16], [17]. Benchmarks comparing pub/sub systems have appeared in industry [18], [19], [20], [21] and academia [22], [23], [24]. However, we identify three common limitations: i) emphasis on stress testing under heavy network traffic rather than emulating real-world IoT workloads [21], [25], [26], [27]; ii) static publisher configurations that overlook heterogeneity in payload sizes, publication rates, and connection stability [22], [28], [29]; and iii) limited representation of large-scale, distributed environments where large numbers of IoT devices publish to applications across various networked servers [30], [28]. Therefore, accurately benchmarking these systems under workloads which accurately characterize the behavior of distributed, heterogeneous IoT deployments remains challenging.

To address these gaps, we present PSMark, a distributed pub/sub benchmarking framework to measure the performance of workloads which accurately represent IoT deployments

containing devices of varying types, payload sizes, publication rates, and network connection stability across both broker-based and brokerless pub/sub systems. PSMark incorporates a modular design which supports the easy addition of new pub/sub protocols, publisher and subscriber implementations, and metrics. The key contributions of this paper are:

- We design and implement PSMark for pub/sub IoT benchmarking which captures IoT deployment-derived behaviors while imposing minimal network and compute overhead, so measurements are not skewed by the framework.
- We define 52 built-in IoT device configurations based on 7 real-world datasets and create 12 built-in workloads comprising domain-specific device deployments.
- We use PSMark to evaluate five popular MQTT brokers (EMQX Open Source, Mochi-MQTT, Mosquitto, NanoMQ, and VerneMQ) and the widely used OpenDDS [31] implementation under PSMark workloads.

The rest of this paper is organized as follows. Section II provides background on pub/sub protocols and benchmarks. Section III details the challenges addressed by PSMark, as well as its architectural design, metrics, and workloads derived from real-world IoT datasets. Section IV provides an overview of the MQTT and DDS systems we benchmark. Section V outlines the experimental benchmarking results. Finally, Section VI concludes the paper and outlines future work.

II. Background and Related Work

Publish/Subscribe Protocols. Pub/sub is a communication paradigm that enables decoupled, asynchronous, event-driven exchange between publishers (data producers) and subscribers (data consumers). Subscribers register a subscription filter (e.g., a topic name) with an event service and receive matching messages sent by publishers to that service. The event service tracks publications and subscriptions, eliminating the need for publishers and subscribers to know about other entities on the network [32]. Two primary protocol families correspond to how the event service is realized: *Broker-based protocols* use a centralized broker to record subscriptions and route matching publications to subscribers; *Peer-to-peer ("brokerless")* protocols rely on distributed, per-node middleware to discover pub/sub entities, manage subscriptions, and route publications.

Pub/sub protocols are widely used in IoT domains; MQTT, AMQP, and DDS are among the most common [33], [22], [9]. MQTT is a standardized, broker-based protocol designed for lightweight machine-to-machine communication in low-bandwidth settings, making it well suited for IoT [6]. AMQP also offers broker-based pub/sub, but prioritizes message safety and reliability, typically incurring higher processing and network overheads [34]. DDS is a fully distributed, brokerless protocol for real-time pub/sub with highly configurable Quality-of-Service (QoS) parameters [7]; it has been adopted across energy grids, healthcare, and manufacturing [10]. Coordination between DDS entities is most often performed via the Real-Time Publish-Subscribe protocol (RTPS) [35], by which the middleware advertises its publishers, subscribers, and QoS parameters. All three protocols use topic-based subscription, where communication occurs over channels identified by keywords known as *topics*.

Related Work. Many pub/sub vendors provide lightweight tools for quick performance checks in simple topologies, such as eMQTT-Bench [36], RTI Perftest [37], and eProsima Fast-RTPS Automated Benchmark [21]. Load testing platforms (e.g., JMeter [25], Gatling [26], and Locust [27]) can be extended with plugins for pub/sub interactions, though they are not tailored for IoT message semantics. Several more comprehensive efforts target pub/sub benchmarking. Longo et al. propose a framework for dynamically deploying multi-broker MQTT systems to study performance in bridged-broker scenarios [38]. Bode et al. analyze four DDS implementations with a cross-vendor DDS benchmarking tool [22]. Zhang et al. introduce a benchmark suite that generates pub/sub clients and workloads with a time-series-based syntax [30].

A few works explore benchmarking pub/sub communication in IoT-specific contexts. Shulka et al. present a benchmark for distributed stream processing that evaluates task-oriented IoT workloads (e.g., model-based prediction, data parsing) based on real IoT data streams [39]. It focuses on end-to-end task benchmarking, and supports MQTT input and output. Zilhão et al. propose a modular pub/sub benchmarking tool for broker-based middleware prioritizing extensibility through swappable modules [28]. They evaluate their middleware through a series of tests sending statically sized messages at a globally consistent rate. Herrnleben et al. design a distributed, IoT-focused, pub/sub benchmark which supports injecting variations to network bandwidth, transmission latency, and packet loss [29]. They then perform an analysis of various static message rates, Quality-of-Service (QoS) settings, and network stability on a one-node, containerized deployment.

Taken together, these tools enable high-level comparisons across different protocol implementations, stress testing of web-based pub/sub systems, or evaluation of general IoT pub/sub behaviors under static payload sizes and constant device connectivity. However, existing tools are often single-protocol, not tailored for pub/sub and IoT environments, do not support measurement of distributed deployments, or do not adequately model dynamic IoT device behaviors (see Table I).

III. PSMark Benchmark

PSMark is, to the best of the authors' knowledge, the first pub/sub benchmark that jointly offers: (i) multi-protocol support, (ii) distributed topology evaluation, (iii) explicit modeling of device heterogeneity and connection stability, and (iv) an open set of IoT-focused workloads derived from real-world datasets designed to be reused and extended by the community. It is publicly available on GitHub [40]. We begin by outlining the design challenges we needed to address.

C-1 Measuring Distributed Topologies. Deployments span rooms, buildings, and cities, often interacting with multiple edge sites [41], [42]. The challenge is to benchmark such geographically dispersed environments and obtain accurate measurements across edge server boundaries.

TABLE I: Comparison of pub/sub benchmarking tools.

Feature	[36]	[37]	[21]	[25]	[26]	[27]	[38]	[22]	[30]	[39]	[28]	[29]	PSMark
Pub/Sub Focused	✓	✓	✓				✓	✓	✓		✓	✓	✓
Multiple Protocol Support				✓	✓	✓			✓		✓	✓	✓
Brokerless Protocol Support		✓		Partial*	Partial*	Partial*	✓		✓				✓
Measures Distributed Topologies							✓	✓		✓		✓	✓
Built-In Workloads						✓	✓			✓		Partial†	✓
IoT Focused										✓		✓	✓
Real-World Device Behaviors										✓		Partial‡	✓
Extensibility				✓	✓	✓					✓	Partial§	✓

* Requires plugin development - † Workload is retired - ‡ Does not model connectivity or variable message attributes - § Cannot extend metrics

C-2 Scaling to Large Populations. Smart-city and IIoT settings involve vast numbers of devices. The challenge is to orchestrate and observe large-scale experiments without the benchmark itself becoming a bottleneck.

C-3 Cross-Paradigm Protocol Coverage. Broker-based and brokerless pub/sub protocols differ fundamentally. The challenge is to support both (with an emphasis on topic-based systems, see Section II) without prescribing specific brokers or configurations, enabling fair, comparable evaluations.

C-4 Capturing Deployment-Derived Workloads. IoT devices publish at heterogeneous rates, with varied payload sizes, and under intermittent connectivity. The challenge is to model and parameterize workloads that faithfully reflect real deployments rather than synthetic stress only.

C-5 Modularity and Extensibility. Pub/sub ecosystems evolve, and publisher/subscriber behavior may include pre-/post-processing (e.g., access control). The challenge is to design a framework that admits new protocols, metrics, and pub/sub client implementations with minimal integration effort.

At its core, PSMark executes deployment-style workload scenarios by instantiating clients (publishers and subscribers) with predefined (user-adjustable) behaviors that exchange messages over the target pub/sub system. PSMark then collects multiple metrics and generates a complete log of results.

A. Distributed and Scalable Architectural Design

To address *C-1* and *C-2*, PSMark coordinates benchmark execution across a configurable set of nodes, analogous to edge servers in an IoT deployment. Each node independently manages: (i) a pool of publishers, which emulate IoT devices by generating the node's assigned workload, and (ii) subscribers, which represent data aggregation points (e.g., databases, applications, AIoT models). By default, each node subscribes to all publications to capture the worst-case fan-out behavior of an IoT deployment. While in most real deployments devices publish from dedicated processors to networked edge servers, emulating one device per compute instance would impose prohibitive orchestration and networking overhead in large scale benchmarking, contradicting *C-2*. PSMark therefore groups publishers per node to leverage local processing power while still ensuring inter-node communication: nodes process messages via externally-facing network interfaces, preserving realistic communication patterns even for brokerless systems.

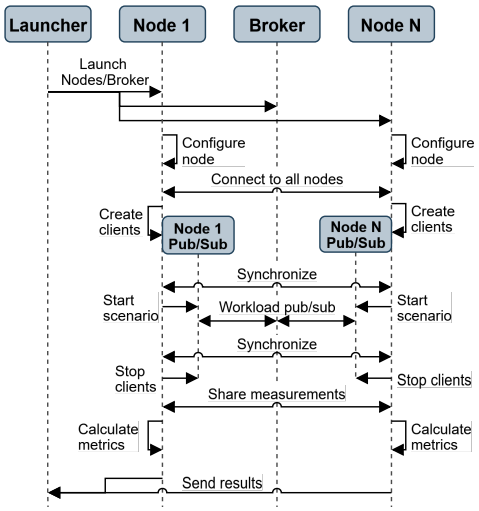


Fig. 1: PSMark execution workflow for a brokered evaluation.

Execution workflow. Figure 1 illustrates the execution workflow of PSMark for a brokered system as an example. First, a *Launcher* process provisions both the broker and the set of PSMark nodes, which in turn configure themselves and establish control-plane connectivity among the set. Next, each node *creates local clients* based on the benchmarking scenario (e.g., a set of publishing smart speakers and cameras in a smart home scenario). At this point, nodes enter a barrier-synchronized lifecycle: (i) *start scenario* only after all peers confirm readiness, ensuring subscribers are active before any publication; (ii) run the *workload pub/sub* phase with no coordination traffic on the data path; (iii) upon completion, hit a second barrier to *stop clients* deterministically; and (iv) *share measurements* needed for per-node and pairwise metrics before computing results locally.

Node Design. Figure 2 overviews each node's architecture and inter-node connections. Two components are primarily responsible for addressing distributed correctness at scale: the *Node Manager* handles intra-node duties (e.g., initialization, verifying connectivity with remote nodes), while the *Lifecycle Manager* coordinates synchronization with remote nodes via a state machine per the workflow presented above. To avoid misclassifying messages as lost, nodes must be synchronized and acknowledge stage completion through the *Lifecycle Manager* before any node may proceed to the next stage (*C-1*). Thus, all subscribers are active before publications begin and remain subscribed until all publications finish. No synchronization

is performed during workload execution to avoid injecting extraneous network overhead, which may skew results.

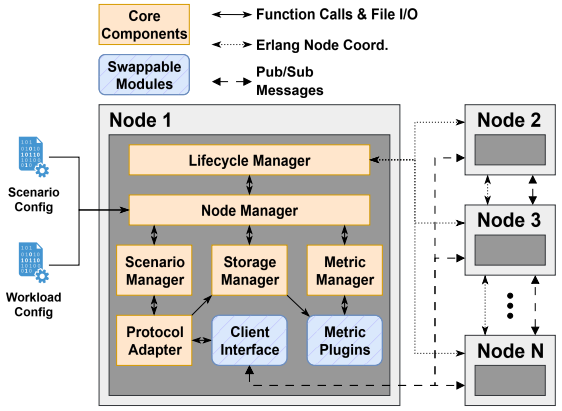


Fig. 2: PSMark node architecture.

PSMark is designed to minimize both runtime and configuration complexity. Nodes are implemented in Erlang/OTP 27. Erlang uses lightweight processes, automatic concurrency and multithreading, and efficient inter-process communication via a persistent TCP connection. This separates coordination from pub/sub traffic and keeps its overhead small compared to the workload itself (*C-1* and *C-2*). Workloads, topologies, and pub/sub systems are configured through declarative specifications; no code changes are required to vary device populations, broker placements, or system parameters.

B. Multi-protocol Adaptation and Extensibility

To future-proof PSMark as an extensible, reusable benchmark for IoT pub/sub evaluation, we adopt a modular architecture that admits new pub/sub protocols, alternative client implementations, and custom metrics (*C-5*) through well-defined interfaces, reducing implementation effort. At the center are *Protocol Adapters* (one per protocol) which translate high-level pub/sub operations into system-specific commands, which can be interpreted by individual pub/sub client interfaces. These adapters also capture measurement events and load system-specific configurations, such as QoS parameters (*C-3*). PSMark has built-in adapters for the MQTT (broker-based) and DDS (brokerless) protocols; adapters for other protocols can be developed for integration into PSMark. Below the adapters, *Client Interfaces* execute pub/sub system-specific commands against concrete pub/sub libraries and can optionally perform pre/post-processing tasks needed to model deployment behavior. For example, to enable content filtering when subscription matching among clients must account for the contents of a message in addition to its topic.

PSMark includes built-in client interfaces for MQTT, using the EMQTT v1.14 library [43] to provide protocol functionality (as we will see in Section IV, this library works with most MQTT broker implementations), and DDS, written using OpenDDS v3.33.0, a well-known, open-source DDS implementation [31], [22]. As OpenDDS primarily supports C++ implementation, we leverage Erlang's Native Implemented Functions (NIFs), which allow C++ code to be executed directly within Erlang processes. Finally, *Metric Plugins* provide a clean extension point for user-defined measurements; Section III-D details built-in metrics and collection.

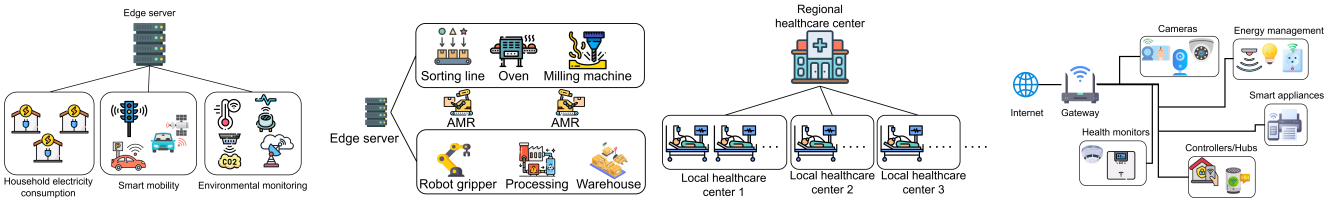
C. Deployment-based Workloads

Within each PSMark node, the *Scenario Manager* (see Figure 2) initializes, coordinates, and terminates pub/sub clients according to a device-centric deployment plan. It issues protocol-agnostic commands (e.g., *connect*, *publish*, *subscribe*) to the underlying clients via the aforementioned *Protocol Adapters*. Workloads are composed of reusable, configurable device definitions. This design models the heterogeneity and complexity of real-world IoT devices [44], [45], allows rapid adjustment of device parameters, and facilitates reuse across user-defined workloads. Each definition specifies message publication rate, payload size, and connection stability. Because a device's payload can vary, users may provide a mean and variance; PSMark then samples payload sizes from a normal distribution. Stability is configured via a *disconnection check period*, *disconnection chance*, *reconnection check period*, and *reconnection chance*. At each check, the publisher running that device's workload disconnects or reconnects according to the given probabilities.

PSMark workloads build on devices; workload definitions specify a composition of devices to model representative IoT deployments[1]. PSMark provides twelve built-in workloads targeting four IoT domains (see Figure 3) with substantial IoT activity and research [2]: smart factory (PSMark-F), smart home (PSMark-HM), smart city (PSMark-C), and smart healthcare (PSMark-HC). We constructed the workloads for each domain by surveying open-source datasets containing real device traces, selecting those with sufficient detail to extract publication frequencies, payload sizes, sleep intervals, and related operational parameters ([46], [47], [48], [49], [50], [51], [52]). Device types, counts, and configuration parameters are derived from the chosen datasets: observed payload structures determine the average and variance of payload lengths; base device counts come from the datasets' deployment context. Because stability is rarely reported, we apply domain-appropriate, workload-wide stability settings frequent enough to exercise disconnect/reconnect behavior (rationales appear in the domain descriptions). Table II summarizes device totals, maximum publish rates (noting that disconnections reduce instantaneous rates), and stability parameters. Each domain includes three workloads scaled at "1x" (base), "2x" and "10x" (device counts multiplied by 2 and 10) respectively, for a total of twelve built-in workloads. Each workload represents the device population of a single edge server; by default, distributed experiments run the full workload per node, though workloads can be split when desired.

PSMark-C: Smart City. PSMark-C aggregates city-scale devices monitoring (i) household electricity use, (ii) traffic and

[1]Users may extend or create workloads from templates to mirror their own deployments (e.g., their specific smart factory).



(a) PSMark-C (b) PSMark-F (c) PSMark-HC (d) PSMark-HM

Fig. 3: PSMark's workload setups: Smart City, Factory, Healthcare Center, and Home.

TABLE II: Summary of PSMark workloads.

Workload	Devices	Msgs/s	Disconnection Rate	Reconnection Rate
PSMark-C	541	109	20% per 5s	50% per 5s
PSMark-C-2x	1,082	218	20% per 5s	50% per 5s
PSMark-C-10x	5,410	1,090	20% per 5s	50% per 5s
PSMark-F	40	590	5% per 1s	80% per 1s
PSMark-F-2x	80	1,180	5% per 1s	80% per 1s
PSMark-F-10x	400	5,900	5% per 1s	80% per 1s
PSMark-HC	24	5	10% per 5s	80% per 5s
PSMark-HC-2x	48	10	10% per 5s	80% per 5s
PSMark-HC-10x	240	50	10% per 5s	80% per 5s
PSMark-HM	20	148	5% per 1s	80% per 1s
PSMark-HM-2x	40	296	5% per 1s	80% per 1s
PSMark-HM-10x	200	1,480	5% per 1s	80% per 1s

mobility, and (iii) environmental air quality. For electricity, we use a smart meter dataset [49] containing 3-minute readings across 100 households, which capture values such as hours of power supply, voltage, and withdrawn current. Mobility devices (vehicle counts, speeds, road occupancy, traffic lights, weather, parking availability, energy consumption, pollution) are derived from a smart mobility dataset [51]. Environmental sensors (humidity, light, NO/NO2, ozone, PM10, CO2, sound, UV, air pressure) follow the Pune Smart City Development Corporation Limited (PSCDCL) testbed [47] with 15-minute intervals. Based on the analysis of these datasets, we include 100 smart meters, 10 devices for each environmental signal, 30 energy-sensing devices, 1 weather station, and 300 traffic sensors per edge server, totaling 541 devices. City devices face less reliable connectivity; we translate this into a stability assessment every 5s with a 20% disconnection chance and 50% reconnection chance. To stress-test protocols under periods of burstiness while maintaining accurate message distributions, we reduce 3-minute and 15-minute intervals to 3s and 15s, respectively, preserving relative rates across device types.

PSMark-F: Smart Factory. We model an assembly line with six machines instrumented by IoT devices, following the setup in [53]. The line includes a sorting station, a multi-processing station (oven and milling), a high-bay warehouse, and a vacuum-gripper robot; Autonomous Mobile Robots (AMRs) support internal material transport [54]. Device parameters are drawn from [50], which provides 10,000 one-minute samples from an industrial machine. We extract six features (e.g., temperature, speed, production quality score, vibration, energy consumption) and model one sensor per feature in the dataset. AMR state and environment ROS [55] sensor readings (obstacle maps, odometry, LiDAR, IMU) are taken from the iV2V testbed [54], [46]. In total, PSMark-F includes 40 devices. Factory devices are assumed stable since they are often stationary and well maintained; we configure a 5% disconnection chance and 80% reconnection chance per second. To better stress pub/sub stacks, we again consider bursty workloads by reducing machine-status intervals from one minute to one second; AMR rates already exceed 10 msgs/s, so the traffic remains AMR-dominated.

PSMark-HC: Smart Healthcare. We model beds requiring continuous monitoring (e.g., ICUs) processed by one server per hospital. The number of monitored beds is based on the average bed count for a medium-sized French hospital, assuming ∼20% require continuous monitoring [56]. Device parameters come from [52] (vitals–heart rate and steps–, activity labels, temperature, humidity, location, device battery). PSMark-HC includes 24 sensors. Hospitals aim for stable connectivity but remain dynamic (patient movement); we therefore choose intermediate stability: every 5s, devices have a 10% disconnection chance and 80% reconnection chance.

PSMark-HM: Smart Home. This scenario models a home IoT analytics platform with controllers (e.g., smart speakers), cameras, energy-management devices (e.g., plugs), health monitors (e.g., smoke detectors, scales), and smart appliances (e.g., printers) based on [48]. Transmission parameters (payload length, rate) are drawn from a real-world deployed testbed [48], [57], [58]. Each home contains 20 devices including Amazon Echo, Belkin Motion Sensor, Belkin Switch, Blipcate BP Meter, Dropcam, HP Printer, iHome PowerPlug, LiFX Bulb, and NEST Smoke Sensor, among others. Home devices are considered stable; we again consider smart home networks and devices reliable and use a 5% disconnection chance and 80% reconnection chance per second.

D. Distributed Measurement and Result Computation

PSMark natively measures pub/sub system performance in terms of network and hardware resource consumption. During execution, each node's *Storage Manager* records message-send/-receive events as well as disconnections and reconnections, persisting them locally in ETS (Erlang's in-memory store) to avoid overhead from networked databases or write-to-disk operations. After a workload completes, the *Storage Manager* ships the necessary aggregates to peer nodes for metric computation, synchronizing it via Mnesia, Erlang's

distributed DBMS. Using ETS and Mnesia ensures efficient data storage and minimizes computational overhead.

The node's *Metric Manager* initializes metric plugins and hardware resource monitors, coordinates metric computation at the end of a run, and logs all events to persistent storage. Each PSMark node reports its own results so that differences in network qualities or machine characteristics do not bias a single global aggregate. In addition, pairwise network metrics are collected among all node-to-node links to maintain accuracy even when some inter-node network paths experience instability (*C-1*). While user-defined metrics can be added as plugins (*C-5*), PSMark includes standard network metrics [59], [60] and hardware-resource consumption metrics:

Message Receipt Throughput: The mean and variance of messages received per second by a node's subscribers. We report throughput from the receiver's perspective, excluding messages that were published but never delivered to that node.

Message Loss: Number of messages published toward a node not received by its matching subscribers. This captures drops in the network, the middleware, or (when present) the broker.

Message Latency: The mean, variance, and P90/95/99 values of the end-to-end time from publisher send to subscriber receipt, measured at the application layer. This includes middleware/broker processing time plus network transit, reflecting IoT deployment experience rather than raw link latency. Publishers include a publication timestamp in the payload; subscribers compute latency without cross-node coordination. PSMark assumes synchronized node clocks prior to execution.

Computational Resources: CPU and memory usage for publishers/subscribers and, when applicable, the broker. To avoid excess overhead, PSMark deterministically designates a single node per scenario to report broker resources. We use Prometheus Node Exporter v1.8.2 [61] on each benchmark node and on the broker's host to query OS-level statistics; this adds negligible network overhead (one query per snapshot).

IV. Pub/Sub Systems

We assess several of the most widely used pub/sub systems using PSMark, including both MQTT and DDS implementations; our benchmark is designed to make such cross-paradigm evaluation systematic and reproducible. We selected open-source MQTT brokers with high popularity (according to GitHub star counts), favorable industry comparisons [62], and demonstrated use in distributed IoT deployments [24]. For DDS, we selected one system based on its popularity (i.e., GitHub star count) and its use of the Adaptive Communication Environment which enables code portability across diverse embedded systems and is widely used across domains such as healthcare, telecommunication, and industrial control [63].

EMQX Open Source [64] (v5.8): Popular broker designed for mass-scale deployments (up to 100 million concurrent MQTT connections per cluster and millisecond-level latency). It uses SQL-based subscription matching to provide highly efficient routing. It is implemented in Erlang/OTP.

Mochi-MQTT [65] (v2.7.9): High-performance broker designed for lightweight operation which can operate as a standalone service or be embedded directly into IoT applications and edge services. It provides each pub/sub client with a dedicated write buffer to avoid one client starving the system. Subscription matching is implemented with a *trie* (a prefix-based search tree optimized for string retrieval). It is implemented in Go.

Eclipse Mosquitto [66] (v2.0.22): Prioritizes a computationally lightweight footprint for use in low power embedded systems. Hashed subscription filter are stored in a topic hierarchy tree to provide more efficient lookup. It is implemented in C.

NanoMQ [67] (v0.24.2): Optimized for multi-core CPUs and implements a built-in scheduler to asynchronously process messaging tasks across multiple threads. It uses a topic trie for subscription matching. It is implemented in C.

VerneMQ [68] (v2.1.1): High-availability broker which emphasizes fault-tolerance through multiple clustered brokers. Subscription matching is also performed with a trie. It is implemented in Erlang/OTP.

OpenDDS [31] (v3.33.0): Popular, open-source, and fully compliant with the DDS specification [7]. It prioritizes operating system portability and interoperability with other DDS systems. It is implemented in C++.

We expect MQTT systems to excel in lightweight messaging at scale. EMQX's mass-scale optimization to favor lower latency and higher throughput under heavy fan-in workloads, Mochi-MQTT to offer minimal message loss and efficient utilization of broker system resources due to its separation of client buffers and the native concurrency provided by the Go language, Mosquitto to provide the lowest broker system resource overhead to ensure its suitability for low power environments, and NanoMQ to show low latency and high throughput in a multi-core testbed. We expect VerneMQ to offer middle-of-the-road performance in a non-clustered, single broker deployment. In contrast, we expect the brokerless DDS stack to incur higher system resource overheads on benchmark nodes (e.g., participant discovery and richer QoS negotiation, which is not offloaded to a broker) but to deliver consistently low latency, and to provide strong reliability under loss when configured to enforce strict QoS requirements.

V. Experimental Evaluation

Experimental Testbeds. We execute PSMark in two environments: a commodity server and a multi-node cluster.

Commodity server: A single, physical host with an Intel Core i7-7700k (4 core) CPU, 32 GB RAM, and running Ubuntu 22.04.1 (kernel 6.8.0-65-generic). The PSMark node, and any MQTT broker under test, run as Docker containers (Engine 28.0.4); nodes use the official *erlang:27-alpine* image as their base; Prometheus Node Exporter v1.8.2 is added to images to capture resource metrics. This setup uses a single PSMark node to emulate small-scale, IoT deployments on commodity hardware using LAN topologies.

Cluster: Two or more physical servers interconnected via a 25 Gbps switch; each server has dual Intel Xeon Gold 5320 CPUs (52 total cores), 384 GB RAM, and hosts a Kubernetes (v1.33.2) cluster spanning multiple VMs (control plane or

workers). All VMs run Debian 12.11 (kernel 6.1.0-37-amd64); containerd v2.1.3 is the container runtime, and Cilium [69] v1.17.6 the Container Network Interface (CNI) plugin. Each worker runs PSMark as a Kubernetes "Pod" so traffic traverses externally facing network interfaces, approximating multi-edge-server deployments. Chrony v4.3 ensures the Kubernetes control-plane node syncs to external Network Time Protocol (NTP) servers and acts as the NTP source for workers. We observe sub-millisecond clock skews across the 16 second sync intervals, ensuring test results are not skewed by clock misalignment. We perform experiments with two, five, and eight PSMark nodes. One server (the "primary") always hosts the broker VM (for MQTT tests) and the control plane VM (16 cores, 64 GB RAM) along with two PSMark VMs (8 cores, 32 GB per worker). Extra servers host only three PSMark VMs. We use one server for the two nodes tests, two servers for the five node tests, and three servers for the eight node tests. Figure 4 shows the five-node configuration.

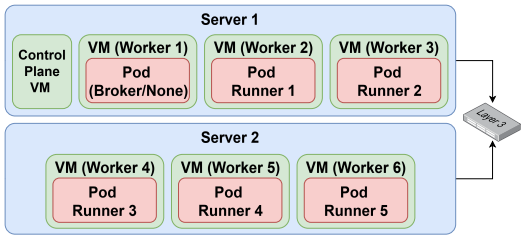


Fig. 4: Five-node testbed configuration.

Pub/sub System Configuration. We benchmark the five MQTT systems using MQTT version 5.0 [6]. To avoid biasing results toward a particular domain or testbed, systems run as close as possible to default configurations; we only change settings necessary for successful execution of PSMark: i) disable EMQX `force_shutdown` setting to prevent intentional crashes on queue-limit hits; ii) allow unauthenticated clients in Mosquitto and VerneMQ (disabled by default); iii) configure OpenDDS for RTPS discovery (rather than the default "InfoRepo" which is marked for deprecation). On the cluster, DDS is configured to use unicast for RTPS, as multicast is known to be unstable in Cilium [70]. These untuned settings represent conservative, "worst-case" expectations. All MQTT brokers are deployed as standalone, self-hosted services. Note that OpenDDS uses TCP by default, ensuring messages traverse the network cards even in the single-node testbed.

Each system's performance is analyzed under low and high QoS requirements. In low QoS mode, MQTT brokers use QoS 0 ("at most once", no delivery guarantees) and OpenDDS uses default QoS (no reliability nor node liveliness guarantees). In high QoS mode, MQTT uses QoS 2 ("exactly once", four-way handshake to ensure message delivery). OpenDDS uses the "RELIABLE" reliability setting (which enforces eventual delivery of messages even should clients temporarily disconnect), the "TRANSIENT" durability setting (with a depth of 5 to ensure late-joining subscribers receive the previous 5 messages on a given topic), and a 1s liveliness lease (which requires DDS entities to send heartbeat messages every second). We denote configurations as X_{LQ} and X_{HQ} (e.g., $EMQX_{LQ}$ or $OpenDDS_{HQ}$).

TABLE III: Five-node communication metrics (low-QoS).

	EMQX$_{LQ}$			*Mochi-MQTT*$_{LQ}$			*Mosquitto*$_{LQ}$		
	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)
C	14.93	691	0.0	11.12	690	0.0	11.96	689	0.1
C-2x	34.85	1381	<0.1	27.17	1381	<0.1	31.65	1328	3.9
C-10x	187.59	6913	<0.1	167.29	6578	4.9	83.79	4648	32.8
F	0.89	2725	<0.1	15.38	2826	<0.1	7.75	2830	<0.1
F-2x	0.94	5493	<0.1	1920.27	4219	25.5	3010.44	3311	41.8
HC	1.32	23	0.0	0.93	23	0.0	1.17	23	0.0
HC-2x	1.31	45	0.0	0.82	45	0.0	1.11	45	0.0
HC-10x	10.74	234	0.0	8.49	233	0.0	11.11	234	0.0
HM	1.02	714	0.0	0.90	713	0.0	1.16	715	0.0
HM-2x	0.96	1432	0.0	0.87	1429	0.0	3.31	1431	0.0
HM-10x	0.95	7266	0.0	0.66	7251	0.0	1.14	7256	0.0

	NanoMQ$_{LQ}$			*VerneMQ*$_{LQ}$			*OpenDDS*$_{LQ}$		
	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)
C	48.60	689	<0.1	9.82	686	0.4	11.00	630	8.6
C-2x	106.64	1379	<0.1	19.77	1333	3.5	17.08	1258	8.8
C-10x	4336.30	5474	20.5	86.75	5351	22.5	N/A	N/A	N/A
F	17.40	2831	<0.1	16.98	2830	<0.1	8.56	2802	1.0
F-2x	5042.69	3994	29.2	296.48	3745	34.0	25990.97	5087	10.0
HC	7.05	23	<0.1	1.14	22	0.0	1.05	21	8.1
HC-2x	12.66	45	<0.1	1.07	45	0.0	2.73	41	8.3
HC-10x	68.77	233	0.2	5.91	234	0.0	21.70	212	8.9
HM	1.47	714	0.0	1.08	714	0.0	0.72	708	0.9
HM-2x	1.52	1430	0.0	0.98	1430	0.0	0.77	1415	0.9
HM-10x	3.16	7262	<0.1	0.81	7259	0.0	N/A	N/A	N/A

Experimental Setup. For each system/testbed, we run all PSMark workloads at their base scaling factors (see Table II) under both low and high QoS configurations. On the cluster, 2x and 10x scales are also run using low QoS. Each experiment consists of 10 minutes of consistent workload traffic, repeated seven times (70 minutes total) for the multi-node testbed, and four times (40 minutes total) for single-node. Reported results are the mean over iterations. Configurations and results are provided in the PSMark repository [40]. In all tables, "N/A" indicates that the pub/sub system could not complete the workload under the given configuration (e.g., due to middleware/broker overload or fault).

A. Multi-Node Experiments

We evaluate the systems on two, five, and eight distributed nodes. For conciseness, we report the five-node results; full results for all topologies are available in [40]. PSMark-F-10x exceeded the capacity of every system in the five-node deployment and is therefore omitted from the analysis below.

Experiment 1 - Workload Network Performance. We first examine communication metrics: latency, throughput, and dropped rate. Due to space constraints, we focus on low-QoS configurations (see Table III and Table IV).

EMQX successfully processes all PSMark workloads with $> 99.9\%$ delivery rate, and is the only system to sustain near-real-time rates on PSMark-F-2x. However, EMQX showed significant latency on all PSMark-C scales (second only to NanoMQ). Because PSMark-C includes an order of magnitude more devices with higher instability than any other workload,

this suggests EMQX performance is more sensitive to the quantity and frequency of client connections rather than the rate of message publication, counter to our initial expectation.

Mochi-MQTT performs unexpectedly strongly on all PSMark-HC and PSMark-HM scales, delivering the lowest latency among MQTT systems (except VerneMQ on PSMark-HC-10x) with no message loss. On PSMark-C-10x it drops fewer messages and achieves higher throughput than Mosquitto, NanoMQ, and VerneMQ, though it trails EMQX. However, Mochi-MQTT struggles on PSMark-F-2x, where the the scenario overwhelmed Mochi-MQTT's message buffers causing high latency and message loss as well as low throughput.

Mosquitto showed average performance overall. It achieves competitive latency values for MQTT on most workload scalings (with only EMQX having lower latency percentiles on PSMark-F), but is easily overloaded: both PSMark-C-10x and PSMark-F-2x show low throughput and >30% loss. Designed for embedded/low-power contexts, Mosquitto trades scale for footprint. Notably, PSMark-HM-10x runs at about half the latency of PSMark-HM-2x, indicating performance does not scale proportionally to client count or message volume.

NanoMQ underperformed expectations, with consistently higher latency and message loss across most workloads. Although it achieves high throughput and comparatively lower loss than Mosquitto an VerneMQ on the demanding PSMark-C-10x and PSMark-F-2x scenarios, it does so with >4s latency—unsuitable for real-time IoT. NanoMQ supports heavy multi-threading, but reaching its potential appears to require intentional, deployment-specific tuning beyond defaults.

VerneMQ behaved as expected in a single-broker setup: competitive on lighter PSMark-HC and PSMark-HM workloads (comparable to EMQX and Mochi-MQTT), and generally lower latency, but with higher message loss on PSMark-C and PSMark-F-2x. We anticipate better performance in multi-broker scenarios, although that is out of the scope of this paper and will be explored as future work.

OpenDDS shows low latency on PSMark-C-2x and the lighter PSMark-HM scales, but exhibits generally higher loss and poor scalability under low QoS settings (e.g., ∼26s latency on PSMark-F-2x; failure on PSMark-C-10x and PSMark-HM-10x). While this is contrary to DDS's strong performance in prior studies [71], [22], these studies configured "RELIABLE" transport, better comparable to the high QoS configuration. As with NanoMQ, OpenDDS seems to benefit strongly from deployment-specific tuning to reach expected performance.

Overall, under low QoS, Mochi-MQTT and VerneMQ obtain the best performance across all three metrics for lighter workloads, while EMQX trades slightly higher latency for far greater reliability in high-intensity IoT deployments. Metrics exhibited high variance across systems/workloads, indicating parameter tuning is required for consistent delivery timings.

Experiment 2 - Hardware Resource Utilization. We next analyze resource usage, again focused on low-QoS configurations. Memory remained between ∼1.2% and 2.0% of the allocated RAM for both benchmark nodes and brokers.

TABLE IV: Five-node latency percentiles (low-QoS).

	$EMQX_{LQ}$			$Mochi\text{-}MQTT_{LQ}$			$Mosquitto_{LQ}$		
	P90	P95	P99	P90	P95	P99	P90	P95	P99
C	42.30	50.28	65.24	31.53	39.34	54.32	34.04	41.27	54.65
F	1.38	1.55	1.93	39.85	46.20	57.79	19.65	25.50	46.75
HC	2.51	3.07	4.41	1.80	2.17	3.25	2.18	2.43	3.27
HM	1.55	1.71	2.01	1.38	1.53	1.80	1.73	1.91	2.32
	$NanoMQ_{LQ}$			$VerneMQ_{LQ}$			$OpenDDS_{LQ}$		
	P90	P95	P99	P90	P95	P99	P90	P95	P99
C	141.64	169.41	219.48	30.62	38.62	53.68	32.35	46.98	76.06
F	44.39	50.49	57.35	40.02	46.82	61.92	26.19	30.81	38.74
HC	12.18	13.83	16.89	2.14	2.55	4.01	2.16	3.00	5.18
HM	2.61	3.08	4.01	1.66	1.84	2.20	1.16	1.34	1.76

TABLE V: Five-node broker/DDS CPU usage (low-QoS).

	$EMQX_{LQ}$	$Mochi_{LQ}$	$Mosquitto_{LQ}$	$NanoMQ_{LQ}$	$VerneMQ_{LQ}$	$OpenDDS_{LQ}$ (Delta)
C	4.13	1.86	1.41	3.80	3.33	2.95
C-2x	7.58	3.07	2.01	7.20	5.35	6.12
C-10x	28.02	14.10	7.60	33.85	25.06	N/A
F	15.25	24.54	12.36	12.92	22.56	3.49
F-2x	20.69	30.47	21.60	25.75	42.03	9.40
HC	0.82	0.51	0.55	0.66	0.50	0.06
HC-2x	1.12	0.44	0.47	0.62	0.63	0.06
HC-10x	1.38	0.91	0.69	1.54	1.05	0.81
HM	6.94	2.49	1.53	3.68	4.00	1.50
HM-2x	10.20	4.60	1.95	5.44	7.10	3.09
HM-10x	25.89	12.09	5.58	12.91	18.63	N/A

Benchmark-node CPU usage across MQTT tests was largely stable (exceeding a 2% delta only for PSMark-F and PSMark-C-10x), as expected given that brokers bear the queuing and routing load. Pub/sub system CPU results appear in Table V. To quantify *OpenDDS* overhead, we take the mean MQTT PSMark node CPU utilization across a workload as a node's baseline under that workload (which excludes broker-side routing/matching). We then compare OpenDDS CPU usage to this baseline. *Mosquitto's* suitability for constrained environments is clear: it consistently uses the least CPU with the exception of 3 workloads which exhibited a <1% delta. *Mochi-MQTT* is also efficient against PSMark-C, PSMark-HC, and PSMark-HM scalings. *EMQX's* power shows the tradeoff of higher CPU usage under lighter workloads. *NanoMQ* and *VerneMQ* were broadly comparable. As expected, OpenDDS adds minimal CPU load for light workloads, with overhead increasing steadily as client counts and publish rates rise.

Experiment 3 - Impact of High QoS Requirements. We now consider the effects of high QoS settings on the workloads' base scaling (see Table VI). All systems incur a notable latency increase, with none able to provide timely transfer under defaults against PSMark-C in a single-broker setup. Mochi-MQTT is the only system with zero message loss under high QoS, slightly improving over its low-QoS results. Other MQTT brokers exhibited some reliability degradation on PSMark-C and PSMark-F due to the increased message counts, processing overheads, and queue holding required to ensure reliable delivery of messages. NanoMQ fails to complete PSMark-F under any test. OpenDDS's high QoS configuration drastically improves message delivery rates over unreliable settings, further highlighting the importance of tuning.

TABLE VI: Five-node communication metrics (high-QoS).

	EMQX$_{HQ}$			*Mochi-MQTT*$_{HQ}$			*Mosquitto*$_{HQ}$		
	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)
C	38.12	588	14.8	31.85	690	0.0	38.69	566	17.8
F	1.86	2729	<0.1	15.28	2830	0.0	53.66	1039	<0.1
HC	5.87	23	0.0	3.36	23	0.0	7.22	22	0.0
HM	1.47	716	0.0	1.40	714	0.0	2.12	710	0.0

	NanoMQ$_{HQ}$			*VerneMQ*$_{HQ}$			*OpenDDS*$_{HQ}$		
	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)
C	113.78	688	<0.1	65.47	565	18.0	45.00	688	<0.1
F	N/A	N/A	N/A	292.09	2633	6.6	9.22	2808	0.6
HC	15.90	22	<0.1	11.83	23	0.0	18.15	22	0.0
HM	2.01	714	0.0	†2.04	†714	†0.0	1.47	712	0.2

†Results represent 5 successful tests.

B. Single-Node Experiments.

We also evaluate the systems on commodity hardware on the single-node testbed. Because smart city and non-base-scaled workloads are infeasible on commodity hardware, we limit our analysis to PSMark-F, PSMark-HC, and PSMark-HM, and compare trends with the multi-node results. The results for low-QoS configurations is shown in Table VII. All systems easily handle the reduced traffic from a single node. MQTT brokers exhibit no loss with uniformly low latency. OpenDDS shows significantly more reliability in a single-node setting, processing workloads with comparable latency to the MQTT brokers with nearly an order of magnitude less drops than it's five-node performance. EMQX exhibited higher latency than in the 5-node environment for PSMark-F, but showed slightly higher reliability overall. Mosquitto maintains the lowest CPU usage among brokers for heavier workloads; but is outperformed on other workloads; EMQX remains the most resource-hungry. Under high QoS, MQTT latency once again increases across the board. NanoMQ and VerneMQ drop $<0.1\%$ of their messages, while all other brokers maintain perfect reliability. OpenDDS shows the largest gains, achieving perfect reliability with latency comparable to its low-QoS runs. Overall, single-node IoT deployments experience few reliability concerns. These results further underscore the need to configure scalable, clustered brokers in multi-node MQTT contexts to allow load balanced data transfer and avoid overwhelming brokers with excessive message frequency.

TABLE VII: One-node communication metrics (low-QoS).

	EMQX$_{LQ}$			*Mochi-MQTT*$_{LQ}$			*Mosquitto*$_{LQ}$		
	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)
F	1.95	568	0.0	2.43	562	0.0	4.91	566	0.0
HC	1.26	5	0.0	0.71	5	0.0	0.64	5	0.0
HM	0.82	143	0.0	0.52	144	0.0	0.47	142	0.0

	NanoMQ$_{LQ}$			*VerneMQ*$_{LQ}$			*OpenDDS*$_{LQ}$		
	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)	Lat (ms)	Tput (msg/s)	Drop (%)
F	2.03	563	<0.1	1.99	566	<0.1	2.69	563	<0.1
HC	1.65	5	0.0	1.29	5	0.0	0.65	5	0.9
HM	0.52	143	0.0	0.70	143	0.0	0.68	144	<0.1

VI. Conclusion

We introduced PSMark, a distributed benchmarking framework for pub/sub systems under deployment-derived IoT workloads. PSMark measures end-to-end behavior while capturing IoT device heterogeneity (publication frequencies, payload sizes, and connection stability) across domains such as smart cities, IIoT, healthcare, and smart homes. Its modular architecture supports custom metrics, additional pub/sub protocols, and deployment-specific client implementations. The PSMark implementation currently includes 12 built-in workloads, derived from 7 datasets that reflect observed device behaviors, 2 protocols (MQTT, DDS), and network and computational resource metrics. The full codebase, including Kubernetes and Docker launchers for easy multi-node and single-node deployments, and all experimental artifacts (configs, logs, and raw results) for reproducibility are available on GitHub [40].

Evaluating six popular pub/sub systems with PSMark has lead to several interesting observations: 1) *Workload composition matters more than raw volume.* Although PSMark-F-2x (80 devices; 1,180 msgs/s) stressed most systems, PSMark-HM-10x (200 devices; 1,480 msgs/s) did not; implicating device mix (rates, payload sizes) rather than message count alone as the dominant driver of performance. This validates PSMark's device-centric workload design over uniform "publishers $\times$ rate" load models. 2) *QoS–throughput trade-offs are steep at scale.* Tightening guarantees (e.g., MQTT QoS 2, DDS reliable + durable + liveness) significantly increased latency and loss (or even led to failures) for most systems on high-volume workloads. Even when latency is not critical, reliability settings must be balanced against system resources and queueing capacity. 3) *Resource efficiency vs. spare capacity varies widely.* Mosquitto consumed the least CPU in several scenarios (up to 78% lower than peers on PSMark-HM-10x) but saturated early. Where messages are independent, sharding across multiple lightweight brokers may be more resource-efficient than a single high-capacity broker. 4) *Default configurations hinder performance but enable generality.* While it was expected that some systems would fail to achieve their full potential under default configurations, others (e.g., EMQX) effectively processed multiple workloads. This is important for hybrid scenarios (e.g., smart cities integrating smart home devices), where tuning for a single domain may not be optimal.

As future work, we plan to (i) add a PSMark control plane for dynamic node orchestration, system configuration, and network impairment emulation; (ii) support automated placement of publisher groups across independent compute instances to better mirror real topologies; (iii) introduce fine-grained topic controls and policy-driven device behaviors; and (iv) extend metrics and protocol coverage, including protocols using content-based subscription which do not require channels to be identified by topics.

Acknowledgment

This work is partially supported by the European Union's Horizon Europe research and innovation actions under grant agreements No. 101168560 (CoEvolution) and No. 101168465 (MEDIATE) and NSF Award #2451803. Experiments were partly carried out using the Grid'5000, supported by Inria, CNRS, RENATER and others (see https://www.grid5000.fr).

REFERENCES

[1] Richard. Miller and Kelli. Washington, *Consumer Use of the Internet and Mobile 2026.*, 10th ed., ser. RKMA Market Research Ebook Series v.4. Richard K. Miller & Associates, 2025.

[2] R. Chataut *et al.*, "Unleashing the power of iot: A comprehensive review of iot applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0," *Sensors*, vol. 23, 2023.

[3] G. Bouloukakis *et al.*, "Unlocking aiot efficiency in the computing continuum - the pandora framework," in *15th Int. Conf. on the IoT*, 2025.

[4] S. I. Siam *et al.*, "Artificial intelligence of things: A survey," *ACM Trans. Sen. Netw.*, vol. 21, 2025.

[5] H. Fu *et al.*, "Aiot: Artificial intelligence and the internet of things for monitoring and prognosis of systems and structures," *IEEE Trans. on Instrumentation and Measurement*, vol. 74, 2025.

[6] OASIS MQTT TC, "Mqtt version 5.0," 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[7] Object Management Group, "OMG Data Distribution Service (DDS)," 2015. [Online]. Available: https://omg.org/spec/DDS

[8] M. Nast *et al.*, "A survey and comparison of publish/subscribe protocols for the industrial internet of things," in *12th Int. Conf. on the IoT*, 2023.

[9] IIoT World, HiveMQ, "Trends and Technology Adoption," 2019. [Online]. Available: https://iiot-world.com/industrial-iot/connected-industry/survey-results-building-iiot-systems-trends-and-technology-adoption/

[10] Object Management Group, "Who's using dds," 2025. [Online]. Available: https://dds-foundation.org/who-is-using-dds-2/

[11] B. Mishra and A. Kertesz, "The use of mqtt in m2m and iot systems: A survey," *IEEE Access*, vol. 8, 2020.

[12] HiveMQ, "Berlex connects traffic signals to the cloud with hivemq," 2020. [Online]. Available: https://hivemq.com/case-studies/berlex/

[13] I. Ahmed *et al.*, "Wireless Communications for the Hospital of the Future: Requirements, Challenges and Solutions," *Int. J. of Wireless Info. Networks*, vol. 27, 2020.

[14] J. Ma *et al.*, "A customizable benchmarking tool for evaluating personalized thermal comfort provisioning in smart spaces using digital twins," *Pervasive Mob. Comput.*, 2025.

[15] N. Papadakis *et al.*, "A context-aware benchmarking approach for scenario-based evaluation of smart devices," in *IoT*, 2025.

[16] A. Hamid *et al.*, "Genaipabench: A benchmark for generative ai-based privacy assistants," *Proc. Priv. Enhancing Technologies*, 2024.

[17] P. Gupta *et al.*, "Smartbench: A benchmark for data management in smart spaces," *Proc. VLDB Endow.*, vol. 13, 2020.

[18] M. Jin, "Open MQTT Benchmarking Comparison," 2023. [Online]. Available: https://emqx.com/blog/open-mqtt-benchmarking-comparison-mqtt-brokers-in-2023

[19] OpenDDS Foundation, "OpenDDS Performance Testing," 2024. [Online]. Available: https://opendds.org/about/performance.html

[20] A. Khizhniak and Y. Goortovoi, "Collection of 20+ MQTT Broker Performance Benchmarks," 2023. [Online]. Available: https://altoroslabs.com/blog/a-collection-of-mqtt-broker-performance-benchmarks-2020-2023

[21] eProsima, "eProsima/benchmarking," 2022. [Online]. Available: https://github.com/eProsima/benchmarking

[22] V. Bode *et al.*, "Systematic Analysis of DDS Implementations," in *MIDDLEWARE*, 2023.

[23] A. Aguiar and R. Morla, "Lessons learned and challenges on benchmarking publish-subscribe iot platforms," in *CPS-IoTBench*, 2019.

[24] H. Koziolek *et al.*, "A Comparison of MQTT Brokers for Distributed IoT Edge Computing," in *Software Architecture: 14th European Conf*, 2020.

[25] The Apache Software Foundation, "Apache jmeter," 2025. [Online]. Available: https://jmeter.apache.org/index.html

[26] Gatling Corp., "Gatling," 2025. [Online]. Available: https://gatling.io/

[27] J. Heyman *et al.*, "Locust," 2025. [Online]. Available: https://locust.io/

[28] L. Zilhão *et al.*, "A Modular Tool for Benchmarking IoT Publish-Subscribe Middleware," in *19th WoWMoM*, 2018.

[29] S. Herrnleben *et al.*, "ComBench: A Benchmarking Framework for Publish/Subscribe Communication Protocols Under Network Limitations," in *Performance Evaluation Methodologies and Tools*, 2021.

[30] K. Zhang *et al.*, "PSBench: A benchmark for content- and topic-based publish/subscribe systems," in *MIDDLEWARE*, 2014.

[31] OpenDDS Foundation, "OpenDDS," 2025. [Online]. Available: https://github.com/OpenDDS/OpenDDS

[32] P. T. Eugster *et al.*, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, 2003.

[33] C. C. Sobin, "A Survey on Architecture, Protocols and Challenges in IoT," *Wireless Personal Communications*, vol. 112, 2020.

[34] OASIS AMQP TC, "Amqp v1.0," 2012. [Online]. Available: https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf

[35] Object Management Group, "DDS Interoperability Wire Protocol," 2022. [Online]. Available: https://omg.org/spec/DDSI-RTPS

[36] EMQ Technologies Inc., "Lightweight mqtt benchmark tool written in erlang," 2025. [Online]. Available: https://github.com/emqx/emqtt-bench

[37] RTI Connext DDS Community, "Rti perftest," 2022. [Online]. Available: https://github.com/rticommunity/rtiperftest

[38] E. Longo *et al.*, "Border: A benchmarking framework for distributed mqtt brokers," *IEEE IoTJ*, vol. 9, 2022.

[39] A. Shukla *et al.*, "RIoTBench: An IoT benchmark for distributed stream processing systems," *CCPE*, vol. 29, 2017.

[40] C. Badolato *et al.*, "PSMark: A distributed pub/sub iot benchmark," 2026. [Online]. Available: https://github.com/DAMSlabUMBC/PSMark

[41] B. Gupta and M. Quamara, "An overview of internet of things (iot): Architectural aspects, challenges, and protocols," *CCPE*, vol. 32, 2020.

[42] W. Yu *et al.*, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, 2018.

[43] EMQ Technologies, "emqtt," 2025. [Online]. Available: https://github.com/emqx/emqtt

[44] A. Chio *et al.*, "Smartspec: A framework to generate customizable, semantics-based smart space datasets," *Pervasive Mob. Comput.*, 2023.

[45] C. Badolato *et al.*, "Your smart home exchanged 3m messages: Defining and analyzing smart device passive mode," in *PerCom*, 2025.

[46] R. Hernangómez *et al.*, "Toward an AI-Enabled Connected Industry: AGV Communication and Sensor Measurement Datasets," *IEEE Communications Magazine*, vol. 62, 2024.

[47] Akshit Mahajan, "Pune Smart City Dataset," 2022. [Online]. Available: https://kaggle.com/datasets/akshman/pune-smartcity-test-dataset

[48] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE TMC*, 2018.

[49] S. Agrawal *et al.*, "High frequency smart meter data from two districts in India (Mathura and Bareilly)," 2021.

[50] Python Developer, "Smart Manufacturing Process Data," 2025. [Online]. Available: https://www.kaggle.com/datasets/programmer3/smart-manufacturing-process-data

[51] Ziya, "Smart Mobility Traffic Dataset," 2025. [Online]. Available: https://kaggle.com/datasets/ziya07/smart-mobility-traffic-dataset

[52] Vinod Kumar, "IoT in Healthcare & Well-being," 2025. [Online]. Available: https://www.kaggle.com/datasets/dcsavinod/iot-in-healthcare-and-well-being

[53] University of Trier IoT Lab, "Fischertechnik Smart Factory," 2025. [Online]. Available: https://www.uni-trier.de/en/index.php?id=69689

[54] R. Hernangómez *et al.*, "Toward an ai-enabled connected industry: Agv communication and sensor measurement datasets," *IEEE Communications Magazine*, vol. 62, 2024.

[55] M. Quigley *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009.

[56] J.-Y. Lefrant *et al.*, "Icu bed capacity during covid-19 pandemic in france: From ephemeral beds to continuous and permanent adaptation," *ACCPM*, 2021.

[57] A. Sivanathan *et al.*, "Characterizing and classifying iot traffic in smart cities and campuses," in *INFOCOM Workshops*, 2017.

[58] R. Kumar *et al.*, "Iot network traffic classification using machine learning algorithms: An experimental analysis," *IEEE IoTJ*, vol. 9, 2022.

[59] M. S. Aslanpour *et al.*, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, vol. 12, 2020.

[60] A. Obiniyi *et al.*, "New innovations in performance analysis of computer networks: A review," *Int. J. of Applied Info. Systems*, vol. 6, 2014.

[61] Prometheus Authors, "Prometheus node exporter," 2025. [Online]. Available: https://github.com/prometheus/node_exporter

[62] F. Wang, "Comparison of Open Source MQTT Brokers," 2025. [Online]. Available: https://www.emqx.com/en/blog/a-comprehensive-comparison-of-open-source-mqtt-brokers-in-2023

[63] DOCGroup, "ACE, TAO, and CIAO Success Stories," 2018. [Online]. Available: https://cs.wm.edu/~dcschmidt/ACE-users.html

[64] EMQ Technologies Inc., "Emqx," 2025. [Online]. Available: https://github.com/emqx/emqx

[65] Mochi MQTT, "Mochi-mqtt server," 2025. [Online]. Available: https://github.com/mochi-mqtt/server

[66] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, 2017.

[67] EMQ Technologies Inc., "Nanomq," 2025. [Online]. Available: https://github.com/nanomq/nanomq

[68] Octavo Labs AG, "VerneMQ: A Distributed MQTT Broker," 2025. [Online]. Available: https://github.com/vernemq/vernemq

[69] Cilium Authors, "Cilium," 2025. [Online]. Available: https://github.com/cilium/cilium

[70] Cilium, "Multicast Support in Cilium (Beta)," 2025. [Online]. Available: https://docs.cilium.io/en/latest/network/multicast/

[71] M. Aartsen *et al.*, "Analyzing Interoperability and Security Overhead of ROS2 DDS Middleware," in *MED*, 2022.