Grenze International Journal of Engineering and Technology, January Issue

Baseline vs. Middleware Resilience: Socket & Broker Communication under Stress in IoT Edge Networks

Manjot Kaur[1] and Daljeet Kaur[2]
[1-2]Department of Computer Science & Engineering, University Institute of Engineering, Chandigarh University, Mohali-140413, Punjab, India
Email: manjotkaurr31@gmail.com daljeet.pawra@gmail.com

***Abstract*— Ensuring resilient communication in IoT Edge networks is critical for applications ranging from industrial automation to defense systems especially under stressed conditions. This study compares two widely used communication paradigms: socket based and broker based systems. We implemented a reproducible testbed using Mininet and Docker following which we introduced realistic network impairments such as packet loss, corruption, reordering and foreign packet injection. Both paradigms were tested with identical payloads, message rates and node topologies to provide a fair comparison. We evaluate performance in terms of end-to-end latency, throughput, reliability and fault tolerance under these impairments. Our results show that socket communication performs well under normal conditions by offering low latency and high throughput but its performance drops sharply in the presence of induced impairments. Broker based systems have slightly higher baseline latency but remain stable under all tested impairments because of asynchronous message delivery, buffering and built-in fault tolerance. Testing confirms that broker based messaging delivers a higher proportion of messages under adverse conditions. This study clarifies the trade-offs to offer practical guidance for selecting communication paradigms in real-world IoT Edge deployments.**

***Index Terms*— IoT Edge Networks · Socket · Broker · Communication · Resilience · Network Impairments · Performance Evaluation.**

I. INTRODUCTION

IoT Edge networks extend traditional cloud-based systems by bringing computation, storage as well as data processing closer to edge devices such as sensors or actuators or embedded controllers. This paradigm reduces end-to-end latency and minimizes bandwidth usage. It also enables real-time decision-making in applications ranging from industrial automation to defense systems. However, edge networks often operate in unpredictable or constrained environments where communication reliability is affected by packet loss, corruption, reordering and foreign packet injection. Socket-based communication provides a low-level direct mechanism with minimal overhead thereby supporting high throughput and low latency in non impaired stable networks. Its simplicity allows control over message handling and timing. However, it lacks built-in fault tolerance or buffering that makes it vulnerable to impairments. Broker-based systems, in contrast, introduce higher level abstractions that enable asynchronous messaging, message queuing and fault tolerance. They reduce programming complexity and improve resilience under network stress but this comes at the cost of higher baseline latency. Prior research has extensively evaluated both paradigms under ideal conditions.

Grenze ID: 01.GIJET.12.1.175

Socket communication is known to excel in point-to-point connections while brokers improve scalability and message reliability. However, few studies systematically quantify the combined impact of multiple network impairments in edge deployments. This leaves practitioners without clear guidance on selecting the appropriate paradigm for stressed networks.

This study addresses this gap by implementing a reproducible IoT Edge testbed. It simulates realistic impairments including packet loss, corruption, re-ordering and foreign packet injection. We compare socket and broker communication across latency, throughput, reliability and fault tolerance. It helped highlight the trade-offs between simplicity, performance and resilience.

By benchmarking both paradigms under controlled stress, our work provides insights for practitioners deploying IoT systems in challenging environments.

II. Related Work

Communication in IoT networks has been studied extensively. Early efforts focused on lightweight protocols to enable resource constrained devices to interact with gateways and servers. MQTT and its variants such as MQTT-S were designed to provide publish-subscribe services over non-reliable wireless networks. This helped in making them particularly attractive for IoT deployments [1,3].

Similarly, CoAP emerged as a RESTful method optimized for constrained devices and networks providing a request-response model that closely resembles HTTP but with significantly lower overhead [4,5]. AMQP on the other hand was introduced to support reliable message queuing and transactional semantics in distributed systems [2]. These protocols establish the foundation for broker based middleware in IoT systems.

Many research studies compared the performance of these protocols under different conditions. Thangavel et al. compared MQTT and CoAP under a common middleware and demonstrated that while both perform well in lightweight scenarios, MQTT offers better scalability in high-throughput environments [6]. Naik compared MQTT, CoAP, AMQP and HTTP highlighting trade-offs in reliability, complexity and scalability thereby emphasizing the importance of selecting communication protocols according to application constraints [7]. Liu et al. compared MQTTbehavior under varying workloads and showed its sensitivity to network dynamics such as high message rates and broker bottlenecks [8].

Yassein et al. compared MQTT and identified several open issues such as resilience to faults and scalability in heterogeneous IoT environments [12]. Beyond protocol level analysis several works have compared broker-based systems in large-scale or stressed deployments. Luckow et al. evaluated distributed message brokers for scalability in data-intensive environments. They showed that broker choice and configuration significantly influence throughput and latency [9]. Mishra and Tyagi compared broker based publish-subscribe systems in IoT concluding that while brokers improve resilience, they often introduce higher baseline latency compared to raw socket communication [10]. Kuryla and Schönwälder evaluated publish-subscribe protocols on constrained networks. They showed that broker overhead can become a limiting factor in ultra-low power deployments [13]. These studies highlight relevance of broker architectures for resilience but also reveal the trade-offs against lightweight socket communication.

At the architectural level, IoT communication is increasingly integrated with edge and fog computing environments. Cirani et al. presented IoT-OAS which is an authorization service for IoT scenarios. They showed the importance of secure broker-based messaging in heterogeneous environments [11]. Zhang and Ansari addressed interoperability issues across IoT communication protocols thereby stressing the need for adaptive middleware in multivendor ecosystems [14]. Morabito et al. explored lightweight virtualization for IoT edge computing showing how container-based deployments (e.g., Docker) facilitate scalable service delivery [15]. These architectural advances have made it possible to systematically evaluate sockets and brokers under realistic edge conditions.

Hence, existing work has provided valuable insights into the design and evaluation of communication protocols and broker systems for IoT. However, most studies evaluate protocols either under idealized conditions or focus on specific performance dimensions such as throughput or energy consumption. Comparative studies between baseline socket communication and broker middleware under stressed and impaired network conditions remain limited.

Particularly, impairments such as packet loss, corruption, reordering and foreign packet injection have not been systematically considered in previous works. This motivates our study by explicitly benchmarking sockets and brokers under controlled impairments in an emulated IoT Edge environment thereby bridging a critical gap between protocol-level analysis and practical deployment.

III. Motivation

The growing deployment of IoT Edge networks in industrial automation, defense systems, and real-time monitoring environments highlights the pressing need for resilient and efficient communication mechanisms. In real-world scenarios, IoT devices operate under unreliable and interference-prone conditions where packet loss, corruption, and reordering can severely degrade system performance. Ensuring reliable data delivery in such stressed environments has direct benefits for safety-critical operations, decision-making accuracy, and system continuity. Previous research on protocols such as MQTT, CoAP, and AMQP has shown the advantages of broker-based middleware in achieving scalability and fault tolerance, while socket-based communication remains unmatched for low-latency, lightweight data exchange. These contrasting attributes motivated us to bridge theoretical insights from prior studies with a practical, reproducible evaluation that tests both paradigms under controlled impairments. By linking real-world reliability concerns with existing communication models, this work aims to identify the most suitable approach for resilient IoT Edge deployments.

IV. Problem Domain

The Internet of Things (IoT) has evolved into a vast ecosystem of interconnected devices generating, transmitting, and processing data across heterogeneous networks. As computation shifts closer to the data source through edge and fog computing, efficient communication becomes a critical enabler of system reliability and performance. Within this broader technological domain, IoT Edge networks operate under constrained bandwidth, intermittent connectivity, and unpredictable interference—conditions that make communication resilience a key concern. Theoretical and practical advancements in distributed computing, middleware design, and networking technologies have enabled varied communication paradigms such as socket-based direct connections and broker-based asynchronous messaging. Each offers distinct trade-offs in latency, fault tolerance, and scalability, forming the technological landscape within which this research is situated.

V. Problem Definition

Although socket-based communication provides low-latency, high-throughput performance in ideal network environments, it lacks built-in mechanisms to handle packet loss, reordering, and corruption—frequent occurrences in real-world IoT Edge conditions. Conversely, broker-based middleware introduces reliability and fault tolerance through message queuing and asynchronous delivery, but at the cost of added latency and resource overhead.Hence, the problem addressed in this work is the absence of systematic, empirical evaluation that quantifies the resilience and performance trade-offs between socket-based and broker-based communication in stressed IoT Edge environments.

VI. Problem Statement

This study aims to evaluate and compare the resilience and performance of socket-based and broker-based communication paradigms under multiple controlled network impairments in IoT Edge environments, thereby identifying the most suitable architecture for reliable and scalable edge deployments.

VII. Methodology

The simulation framework was designed to imitate a heterogeneous IoT Edge network consisting of diverse nodes such as gateways, embedded controllers, and sensor/actuator devices. Each node is modeled as an independent software process capable of generating, transmitting and receiving data streams in real time. Two distinct communication architectures were compared: a socket-based system and a broker-based system. The evaluation focused on their behavior under both ideal as well as impaired network conditions.

A. Socket based Communication

In the socket-based communication system, nodes were arranged in a partial mesh topology. Each node maintained direct TCP connections with a predefined subset of peers thereby enabling continuous point-to-point communication patterns for data generation and exchange. This architecture operated without reliance on a central coordinator thereby embodying a decentralized model. The advantage of this setup lies in its low latency under ideal conditions as data traverses a minimal path between endpoints. However, scalability becomes a concern as the number of nodes increase as each node must manage multiple concurrent TCP sessions to maintain connectivity across the network.

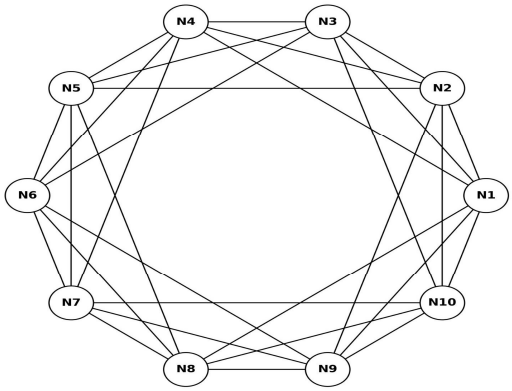

Fig. 1. Socket-Based IoT Edge Network Topology

B. Broker based Communication

In the broker-based communication system, a central RabbitMQ broker served as the intermediary for message exchange. Nodes connected to the broker over TCP. They communicated using a publish-subscribe paradigm. Publishers sent data to specific topics while subscribers automatically received relevant messages. This decoupling of senders and receivers simplified connection management. This added message queuing, delivery acknowledgments and fault recovery. The broker was deployed inside a Docker container to ensure isolation, reproducibility and flexible configuration within the emulated network.

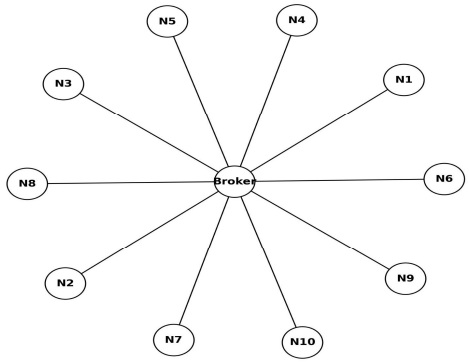

Fig. 2. Broker-Based IoT Edge Network Topology

C. Network Impairments with Mininet tc/netem

Both architectures were deployed within Mininet. It is a lightweight network emulator that provides virtual hosts, switches and links in a controlled and reproducible environment. Mininet leverages Linux namespaces and virtual Ethernet pairs (veth pairs) to isolate each virtual host. This gives it a dedicated network stack, routing table and interface configuration. This allowed the simulated IoT devices and edge nodes to behave as independent physical entities while running on a single host machine. Communication traffic between hosts is forwarded through Mininet virtual switch fabric implemented using Open vSwitch (OVS). This ensured that packet level behaviors such as queuing, retransmissions and congestion were realistically represented. As a result, dynamics of real-world IoT networks including buffer-ing delays and bandwidth constraints were reproduced. The choice of Mininet provided scalability: a 10-node topology was instantiated rapidly on limited hardware thereby eliminating the need for a distributed testbed.

D. Network Impairments

After baseline experiments under ideal conditions, impairments were introduced using Linux tc netem. The following adverse conditions were applied uniformly across inter-host links:–

- Packet loss: Configured at incremental rates to emulate unreliable wireless links.

- Packet corruption: Introduced bit-level errors in transit to test error handling.
- Packet reordering: Applied to simulate out-of-order delivery common in congested or multi-path networks.
- Foreign packet injection: Simulated external traffic to test resilience against interference.

These impairments reflect common challenges in IoT edge networks as interference, congestion and adversarial actions can degrade communication quality.

VIII. Results and Analysis

Each node logged message timestamps, delivery counts and retransmission events. Metrics are computed as follows:

- Latency: Measured as end-to-end delay between message generation and delivery
- Throughput: Data successfully delivered per unit time.

Logs were collected centrally and aggregated for analysis. Each experiment was repeated ten times and results were averaged to account for stochastic variation. This methodology provides a reproducible framework for benchmarking socket based and broker-based communication systems. By combining Mininet, Docker and controlled impairments, we ensured that both architectures were tested under identical and realistic conditions thereby allowing for a fair assessment of their resilience for practical deployments. This section presents the empirical findings from the comparative evaluation of socket-based and broker-based communication systems in an IoT Edge network environment.

The results are organized into three parts: baseline performance without impairments, performance under induced impairments and an analytical discussion of the implications. All metrics are collected from multiple experimental runs and averaged values are reported.

A. Baseline Performance

In the baseline scenario consisting of ten interconnected hosts without any impairments applied, the two architectures demonstrated distinct performance characteristics:

- Latency: The socket-based system achieved a mean end-to-end latency of approximately 0.5 ms whereas the broker-based system achieved a significantly higher latency of 2.1 ms. This overhead in the broker setup can be attributed to queuing delays, internal routing of messages and pub lisher–subscriber mediation.
- Throughput: The socket-based architecture achieved an average throughput of about 2500 messages per second while the broker-based system achieved approximately 2200 messages per second. The lower throughput in the broker case can be attributed to internal queuing, persistence and acknowledgment mechanisms.

While these results clearly favor sockets in low-scale deployments, scalability tests highlighted diverging trends. As the number of nodes increases beyond 10, the latency of sockets increases exponentially as each node is forced to maintain multiple concurrent TCP sessions. This increased connection management overhead and retransmission events.– The latency of brokers remained relatively stable as each node only needed to maintain a single connection to the central broker.– The throughput of sockets degraded rapidly with growing network size whereas the throughput of brokers decreased only marginally due to the centralization of connection management. From a IoT/Edge deployment standpoint, this suggests that socket-based communication is advantageous in small and stable teams (e.g., a few units ex changing time-critical updates) but brokers provide better resilience when scaling to larger teams.

B. Impact of Network Impairments

To reflect stressed environments (e.g., radio interference, adversarial jamming, mobility-driven disruption) impairments were introduced incrementally using tc netem. tc netem stands for Traffic Control Network Emulator. It's a Linux kernel feature (part of the tc traffic control command suite) that lets you simulate real network conditions like delay, packet loss, corruption, and reordering without needing special hardware. The following subsections present the results under specific impairment types.

Packet Loss

- Sockets: With only 5% packet loss latency increased significantly (2 ms → 22 ms). Throughput dropped below 3000 messages per second. TCP's retransmission mechanism and acknowledgments caused repeated retrans missions thereby reducing efficiency.
- Brokers: Latency was relatively stable (2.8 ms → 4.6 ms) while throughput remained above 7800 messages per second. The broker absorbed packet loss through asynchronous delivery, buffering and retransmission at the middle ware layer thereby avoiding reducing efficiency.

TABLE I. IMPACT OF PACKET LOSS ON SOCKET AND BROKER SYSTEMS

Loss (%)	Sockets Latency (m/s)	Brokers Latency (ms)	Socket Throughput (msg/s)	Socket Throughput (msg/s)
0	0.5	2.1	2500	2200
1	1.6	2.4	2000	2100
5	8.0	4.6	700	1850
10	20.0	7.5	300	1500
20	60.0	15.0	90	900

Packet Reordering

- Sockets: At7%reordering latency increased significantly to 26 ms. Throughput dropped to 2500 messages per second. The TCP ordered delivery mechanism forced packets to wait until the missing segment arrived thereby reducing delivery.
- Brokers: Latency remained under 5 ms and throughput was 7500–8000 messages per second. Message sequence tracking at the broker avoided reducing delivery thereby enabling out-of-order handling at the middleware level.

TABLE II. IMPACT OF PACKET REORDERING ON SOCKET AND BROKER SYSTEMS

Reordering (%)	Socket Latency (ms)	Socket Throughput (msg/s)	Broker Latency (m/s)	Broker Throughput {msg/s)
0	0.5	2500	2.1	2200
3	12	2800	3.2	8000
5	18	2600	3.8	7600
7	26	2500	4.5	7750
10	38	2200	5.2	7400

Packet Corruption

- Sockets: At 5% corruption latency increased significantly to 24 ms. Throughput dropped below 2500 messages per second. TCP drops and retransmits corrupted packets thereby increasing congestion.
- Brokers: Latency remained under 5 ms and throughput was 8000 messages per second. The broker discarded corrupted packets and retransmitted them without heavily penalizing end-to-end performance thereby avoiding congestion.

TABLE III. IMPACT OF PACKET CORRUPTION ON SOCKET AND BROKER SYSTEMS

Reordering (%)	Socket Latency (ms)	Socket Throughput (msg/s)	Broker Latency (m/s)	Broker Throughput {msg/s)
0	0.5	2500	2.1	2200
3	14	2300	3.0	7950
5	24	2500	4.5	8000
7	35	2100	5.0	7800
10	48	1800	5.5	7600

Foreign Packet Injection

- Sockets: When foreign packets were injected throughput dropped below 2000 messages per second. Nodes wasted CPU cycles processing irrelevant traffic which delayed valid packet handling.
- Brokers: The central broker efficiently filtered foreign packets prioritizing valid packet handling. Latency remained below 5 ms. Throughput remained above 6800 messages per second. CPU utilization of the broker spiked to about 55%.

TABLE IV. IMPACT OF FOREIGN PACKET INJECTION ON SOCKET AND BROKER SYSTEMS

Reordering (%)	Socket Latency (ms)	Socket Throughput (msg/s)	Broker Latency (m/s)	Broker Throughput {msg/s)	CPU (%)
0	0.5	2500	2.1	2200	15
2	10	2300	2.5	7500	25
5	18	2100	3.2	7400	35
7	28	2000	4.5	7000	50
10	40	1800	4.8	6800	55

IX. COMPARISON OF RESULTS

The experimental results underline the architectural trade-offs between socket based and broker-based communication:

- Sockets excel in minimal latency and throughput under small and non impairment environments. This makes them suitable for specialized IoT operations where only a few nodes exchange highly time-sensitive data (e.g., local control loops in industrial automation or sensor fusion at the edge).
- Brokers excel at scale and impairment. They maintained stable latency (<5 ms), high throughput (>7000 messages per second) and resilience against packet loss, corruption, reordering, and injection. This resilience is particularly valuable in IoT deployments characterized by heterogeneous devices, wireless interference and adversarial traffic.

Slightly higher CPU and memory utilization along with added queuing latency are the trade-offs. However, these costs can be ignored compared to the benefits of resilience and scalability in large-scale IoT environments. Ultimately, socket-based systems remain advantageous for ultra-low-latency and small-scale topologies but broker-based systems emerge as the preferred design choice for practical deployments.

X. Conclusion

This work presented a comprehensive comparative evaluation of socket-based and broker-based This work presented a comparative evaluation of socket-based and broker-based communication paradigms for IoT Edge networks under stressed and impaired conditions. Using a reproducible testbed built on Mininet and Docker and systematically inducing impairments such as packet loss, corruption, reordering, and foreign packet injection, we characterized the strengths and weaknesses of both architectures with quantitative analysis. The results showed that socket-based communication provides superior latency and throughput in small-scale topologies where impairments are minimal. However, its performance degrades sharply with scale and in the presence of adverse conditions thereby limiting its applicability in dynamic or interference prone environments. In contrast, broker-based messaging systems incur modest baseline overhead. They consistently outperform sockets under stress. They offer better scalability, resilience against impairments and stable performance across diverse IoT and edge scenarios. From a practical standpoint, these findings suggest that socket-based systems are good for tightly coupled and ultra-low-latency tasks (e.g., sensor fusion between a small set of devices) whereas broker-based systems are the prefer able choice for resilient and scalable communication in heterogeneous IoT deployments.

References

[1] Stanford-Clark, A., Truong, H.L.: MQTT for sensor networks (MQTT-SN) protocol specification. Int. J. Sensor Networks, 13(3), 171–180 (2013)
[2] AMQP Working Group: Advanced Message Queuing Protocol (AMQP) Version 1.0. OASIS (2012)
[3] Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S—A publish/subscribe protocol for wireless sensor networks. In: IEEE COMSWARE, pp. 791–798 (2008)
[4] Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252, IETF (2014)
[5] Bormann, C., Castellani, A.P., Shelby, Z.: CoAP: An application protocol for billions of tiny internet nodes. IEEE Internet Comput., 16(2), 62–67 (2012)
[6] Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y.: Performance evaluation of MQTT and CoAP via a common middleware. In: IEEE ISSNIP, pp. 1–6 (2014)
[7] Naik, N.: Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: IEEE ISSE, pp. 1–7 (2017)
[8] Liu, J., Zhao, M., Li, W., Zhang, T., Yu, H.: MQTT performance evaluation and analysis in IoT. In: IEEE ICC, pp. 1–6 (2019)
[9] Luckow, A., Manhardt, F., Emmerich, W.: Evaluation of message brokers for scalable distributed stream processing. In: ACM LSDB, pp. 1–6 (2015)
[10] Mishra, A., Tyagi, A.: Performance analysis of broker-based publish–subscribe protocols for IoT applications. J. Ambient Intell. Humaniz. Comput., 11(11), 4725 4738 (2020)
[11] Cirani, S., Picone, M., Gonizzi, P., Veltri, L., Ferrari, G.: IoT-OAS: An OAuth based authorization service for IoT scenarios. IEEE Sensors J., 15(2), 1224–1234 (2014)
[12] Yassein, M.B., Shatnawi, M.Q., Aljawarneh, S.: Internet of Things: Survey and open issues of MQTT protocol. In: IEEE FiCloud, pp. 1–6 (2016)
[13] Kuryla, S., Schönwälder, J.: Evaluation of publish–subscribe protocols for use on constrained networks. In: IEEE NOMS, pp. 1–9 (2015)
[14] Zhang, T., Ansari, N.: On protocol interoperability in IoT systems. IEEE Internet Things J., 4(5), 1888–1898 (2017)
[15] Morabito, R., Cozzolino, V., Ding, A.Y., Beijar, N., Ott, J.: Consolidate IoT edge computing with lightweight virtualization. IEEE Netw., 32(1), 102–111 (2018)