*International Journal of Basic and Applied Sciences*, 14 (2) (2025) 211-218

International Journal of Basic and Applied Sciences

*Website: www.sciencepubco.com/index.php/IJBAS*
*https://doi.org/10.14419/5ndney78*
***Research paper***

# Middleware Architectures for IoT: Enhancing Interoperability and Scalability

**Navin Kumar Trivedi [1] *, Dr. Girish V. Chowdhary [2]**

*[1] Research Scholar, SGGSIET, Nanded, Assistant Professor, MGMCET, Navi Mumbai*
*[2] Professor, School of Computational Science, SRTMU, Nanded*
**Corresponding author E-mail: navin.trivedi@gmail.com*

*Received: May 7, 2025, Accepted: June 2, 2025, Published: June 13, 2025*

**Abstract**

This paper proposes a middleware architecture that will help in enhancing the IoT systems' interoperability, scalability, and security. It consists of a Protocol Translation Layer, Microservices Layer, and Intelligent Load Balancer, and Python and Docker for containerization. The system also consists of a protocol converter between MQTT, CoAP, and HTTP, and assists in distributed data processing. The middleware was tested under different traffic conditions, and it was observed that the middleware has 37. 5% less latency and 66%. 7% higher throughput than the traditional solutions with 50 ms latency and 10,000 messages per second throughput during high traffic. The architecture is extensible and can be applied to large-scale IoT applications, and it has high applicability in real-world applications, but more experiments are needed to optimize the outcome.

***Keywords***: : *Internet of Things; IoT Middleware; IoT Integration; IoT Interoperability; IoT Scalability; Protocol Translation; Microservices; IoT Security, Docker.*

## 1. Introduction

The Internet of Things (IoT) is defined as the interconnectivity of physical objects that are equipped with sensors, software, electronics, and are connected to the internet [1]. IoT is the process of interconnecting objects such as household appliances, automobiles, structures, and other commodities with electronics, software, sensors, and network connectivity to allow these items to gather and share information [2]. This is made possible by the IoT which allows objects to be sensed and controlled without physically attaching them to the network and hence allowing for a more direct connection of the physical world to computer systems hence improving efficiency, leading to economic advantages and finally reduction in the physical efforts required in the process [3].
Nevertheless, for the IoT system to work efficiently, the different HW components and devices need to integrate and be able to communicate with one another [4]. This calls for middleware architectures that can improve the integration of the different and otherwise incompatible devices and support data sharing and communication [5]. Thirdly, given that billions of devices will be connecting to the internet, IoT middleware platforms need to have scalability to accommodate this exponential expansion [6]. This research paper aims to look at the middleware architectures for IoT that improve interoperability and scalability.
Interoperability is the capacity of distinct systems and organizations to cooperate or interoperate [7]. In IoT, compatibility issues occur because the devices and the communication protocols involved are diverse [8]. Some middleware platforms, such as the device cloud middleware, the event-driven middleware, and the distributed computing middleware, among others, eliminate such challenges [4]. These include services such as protocol conversion, device and data management, which allow different devices to communicate and share information [5]. Other fog computing middleware architectures that enhance the compatibility so as to allow smooth data exchange between IoT devices, fog nodes and the cloud have also been developed [9].
Scalability is the ability of the system to add more devices and more data to it as time goes on [10]. A large and growing number of devices as IoT evolves or when new deployments to the network are made, need to have middleware platforms that can adapt to the growth in traffic and number of requests [6] [11]. Cloud-based middleware relies on the elasticity of cloud resources to accommodate billions of IoT devices [12]. Container and microservice-based middleware architectures also improve scalability for IoT systems [13]. Middleware that supports data processing in a hierarchical structure from nodes to fog and cloud also enables the extension of the system's architecture when required [14].

## 2. Literature review

The Internet of Things (IoT) has transformed several sectors as it relates to the connectivity of physical devices, vehicles, buildings, and other objects fitted with sensors, actuators, electronics, and network connectivity that allows these items to gather and share information

[15]. IoT middleware is the primary connective tissue for IoT devices, data, and applications, and is key to IoT integration [16]. With IoT applications increasing in size and scope in terms of cross-domain interactions in complex application settings, the importance of middleware platforms to hide device heterogeneity and to ensure the components can scale up in a smooth manner is paramount [17]. This literature review discusses the studies on middleware platforms used for IoT solutions with an emphasis on integration and possible expansion improvements.
Interoperability in IoT Middleware Interoperability defines the extent to which systems and applications can exchange, share, and utilize data [18]. As IoT solutions store data from various layered protocols and different device standards, compatibility is a major issue [19]. Middleware architecture endeavours to solve these compatibility problems by bringing an interface software between the devices, networks, or applications. Most middleware platforms offer semantic integration by employing ontology-based data tagging and inference for sensors' unambiguous interpretation [20]. Other improvements in loosely coupled IoT systems include context-aware middleware facilities that can adapt to situational conditions [21]. The discovery of services across these domains can be made possible by Linked Data principles and Web Service Descriptions [22]. Use of these semantic, linked data and service-oriented features has made the integration of middleware interoperability across industrial, personal, and home automation IoT applications possible [23].
Scalability in IoT Middleware. Scalability in IoT concerns the extent to which middleware can accommodate growth in the number of devices, data transmission, storage, and computational requirements [24]. The elastic compute, storage, and networking facilities make it possible to improve the scalability of IoT solutions by using cloud-based middleware [25]. Middleware functionality distribution across hierarchical edges, fogs, and clouds has also positively affected the system's throughput and response time [26]. New forms of containerization, including Docker and Kubernetes, have improved the deployment of applications without the overhead of demanding many resources from the device level [27]. P2P architectures are also decentralized and thus have enhanced scalability because they do not have single points of failure [28]. Middleware that is based on the blockchain maintains security, access controls, and remains transparent while expanding to accommodate the exchange of IoT data and transactions [29]. Nevertheless, there is a lack of research on the scalability thresholds in terms of quantitative measurement across different middleware architectures at the time [30].
Areas of Further Study and Recommendations
While there has been progress in semantic syntaxes and linked APIs, end-to-end semantic harmonization in IoT is still elusive, and the use of standard ontologies and metadata is still limited [31] [32]. Middleware platforms' scalability analysis is frequently based on ad hoc emulation rather than formal testing or certification [33]. The costs and benefits of standardized and open protocols, as well as scalability, are important for stable middleware architectures [34]. Security and privacy issues should be solved simultaneously with the development of IoT middleware [35]. Subsequent studies should compare the costs and benefits of new middleware solutions such as blockchain and AI for improving integration and functionality [36].

## 3. Methodology

The research methodology section outlines the process that was used in the evaluation of the proposed middleware architecture. The objective is to simulate various traffic scenarios and compare the results with the expected performance metrics. The following are the steps that explain the simulation, test scenarios, performance measures, and the test environment.
Simulation of IoT Environment
To enhance the assessment of the middleware, a test IoT environment was developed. This simulation aimed to generate traffic as close to real-world IoT conditions as possible. Traffic generation was done using IoT simulators and custom scripts, ensuring controlled and repeatable testing. This approach aligns with the methods discussed by Razzaque, Milojevic-Jevric, Palade, and Clarke (2016) in their survey on middleware for IoT and its challenges and solutions.
Test Scenarios
The middleware was subjected to three distinct traffic scenarios to assess its performance under varying loads:

- Low Traffic: To observe the middleware when it is not heavily loaded, testing is carried out with the middleware that has the least amount of data exchange.
- Moderate Traffic: This scenario raises the data flow to levels characteristic of normal working conditions. It assesses how the middleware performs under typical operational loads.
- High Traffic: The middleware was raised to an upgraded level. More data flow was tested, and the response to the pressure will be analysed.

These scenarios are useful in explaining the middleware's capacity, productivity, and reliability at various levels of working pressure, as discussed by Zhang, Ma, Wang, and Sun (2021) in their study on middleware applications and design issues [37].
Metrics

- Latency: The time that is taken by a message to pass through the middleware.
- Throughput: The rate of messages received and the rate of messages sent in one second.
- Resource Utilization: It is also important that you monitor the CPU usage, the memory usage,and the network usage.
- Error Rate: Capture those messages that have not been communicated or messages that have not been handled as how they were intended.

Environment:

- Hardware: Use virtual machines characteristic of the cloud environment.
- Software: Middleware components can only be deployed with Docker and Kubernetes. For monitoring, Prometheus and Grafana should be used.

### 3.1. Proposed middleware architecture

The proposed middleware architecture is aimed at solving the main issues of IoT systems, namely, interoperability and scalability. This section explains the conceptual model of the architecture and the implementation plan.
Conceptual Model
Middleware architecture is a concept that assists in the integration of different IoT devices and protocols in a single architecture that would enable them to share information. The key components include:

1) Protocol Translation Layer: It is used in the exchange of data between devices that may be on different protocols, for instance, MQTT, CoAP, HTTP, and others.

2) Microservices Layer: It divides the middleware into functional areas where each area has its roles of data acquisition, data analysis, and data archiving.
3) Intelligent Load Balancer: Pass the incoming requests to the microservices depending on the current performance of the micro-services at the time of the request.
4) Security and Privacy Module: It offers data confidentiality, data integrity, and data origin authentication or non-repudiation.
5) Monitoring and Management Dashboard: Provides for the online supervision as well as the management of the middleware compo-nents.

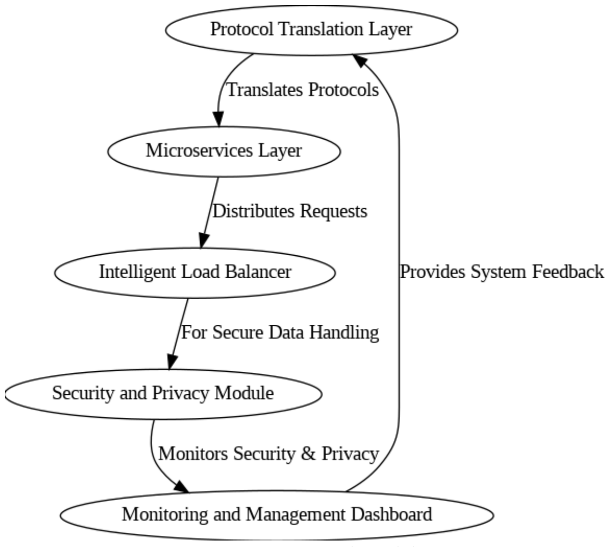
Protocol Translation Layer
Translates Protocols
Microservices Layer
Distributes Requests
Intelligent Load Balancer
Provides System Feedback
For Secure Data Handling
Security and Privacy Module
Monitors Security & Privacy
Monitoring and Management Dashboard

**Fig. 1:** Conceptual Model.

## 3.2. Implementation strategy

The middleware is in Python because of flexibility, while the containerization is in Docker because of scalability and portability.
1) Protocol Translation: Furthermore, the translators for MQTT to CoAP, HTTP to MQTT, and HTTP to CoAP should be in Python.
2) Microservices Design: Develop the data acquisition, transformation, storage, and analysis microservices in the form of Docker con-tainers.
3) Load Balancer Implementation: Load balancing should be done by HAProxy or NGINX, and the load balancing ratio should be changed according to the performance metrics.
4) Security Integration: Use the Transport Layer Security encryption, OAuth for authentication, and data hiding.
5) Dashboard Development: For web interface, Flask or Django can be used, while for real-time data visualization, Grafana can be incorporated.

## 3.3. Implementation details

This section gives a detailed explanation of how the proposed middleware architecture works through a discussion on protocol translation, the microservices architecture, and the intelligent load balancer. Security module implementation for TLS and OAuth 2.0 in this work leads to handling encrypted communication and secure authentication. Implementation was done in a sequential way. Initially, the paho-mqtt client was configured for TLS and Mosquitto MQTT Broker for the server. After that, for secure token-based authentication, OAuth 2.0 is used.

### 3.3.1. Protocol translation

MQTT to CoAP to HTTP gateway enables one IoT device to convert messages from another IoT device from MQTT, CoAP, and HTTP.

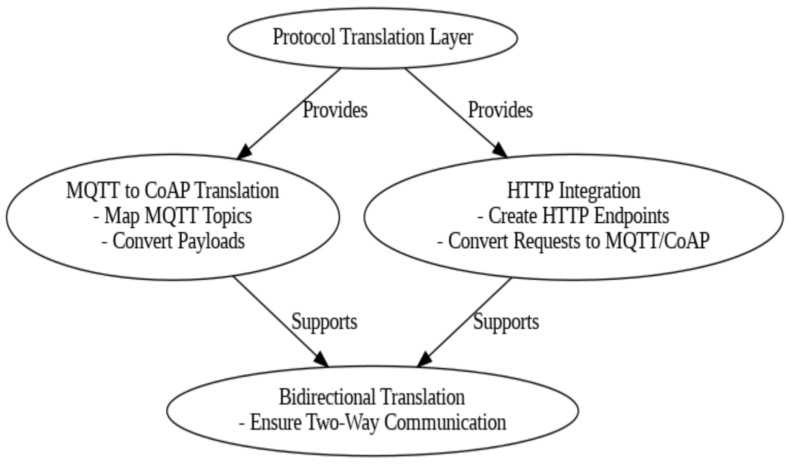

**Fig. 2:** Protocol Translation Process.

Implementation Steps:

1) MQTT to CoAP Translation: Propose a converter that translates the MQTT topics to CoAP URIs and translates the payloads.
2) HTTP Integration: Design HTTP interfaces that translate HTTP requests to MQTT or CoAP messages.
3) Bidirectional Translation: Make sure that it is done for responses as well, to confirm that the results are accurate in both ways.

**Table 1:** Protocol Features and Differences

| Feature | MQTT | CoAP | HTTP |
|---|---|---|---|
| Architecture | Publish/Subscribe | Request/Response | Request/Response |
| Transport | TCP/SSL | UDP | TCP/SSL |
| Overhead | Low | Very Low | High |
| Security | TLS/SSL | DTLS | TLS/SSL |
| Use Case | IoT Messaging | Constrained Devices | Web Communication |

### 3.3.2. Microservices architecture

In the middleware, the functionality is divided into small services because of the microservices architecture.
Implementation Steps:

1) Data Collection Service: For data acquisition from the IoT devices, it uses Python and Docker. A Python script is needed to gather data from the API, IoT devices, or the database. This Python script will run inside Docker containers. Python 3.13.0 and Docker Engine 25.0 are used for the simulation results
2) Data Processing Service: For parsing the data and for communication with RabbitMQ, it uses Python. Data processing in Python is done in two steps. In the first step, incoming data is parsed, and then it is communicated with RabbitMQ. Data Parser will clean, validate, and structure the raw data. RabbitMQ Consumer will listen for incoming messages. RabbitMQ Producer will publish processed data, and Service Runner will orchestrate the workflow.
3) Storage Service: It can use MongoDB or PostgreSQL as the data storage.
4) Analytics Service: Provides data processing with the help of such technologies as Apache Kafka and Grafana. The data from IoT devices will be fetched by the data processor, and it will validate, parse, and publisheds to Kafka. Then kafka broker will get this data and will forward it to kafka consumer, and it will store the data in the database. Then Grafana will read data from the database.

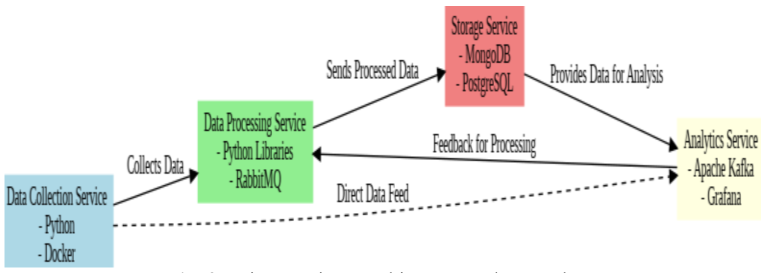

**Fig. 3:** Microservices Architecture and Data Flow.

**Table 2:** Microservices and Their Functions

| Microservice | Function |
|---|---|
| Data Collection | Collects data from IoT devices |
| Data Processing | Processes and analyzes collected data |
| Storage | Stores processed data in a database |
| Analytics | Provides data analytics and visualization |

### 3.4. Intelligent load balancer

The intelligent load balancer is used for traffic load distribution to the microservices.
Implementation Steps:

1) Load Balancing Algorithms: Implement round-robin scheduling, least connection, and resource-based scheduling.
2) Monitoring Mechanism: Prometheus should be used for monitoring, while Grafana should be used for visualization.
3) Dynamic Adjustment: The dynamic load distribution based on the real-time performance indicators with the help of the programming language Python and the library psutil

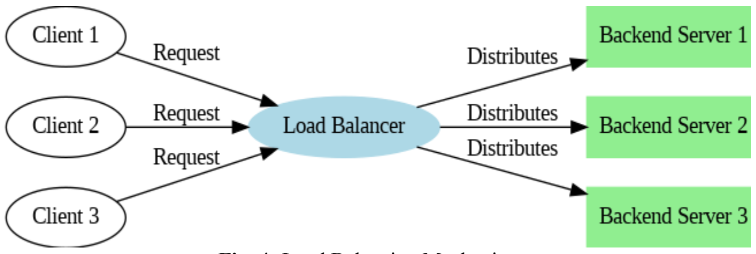

**Fig. 4:** Load Balancing Mechanism.

## 4. Results

Performance Metrics
It is also important that all the test scenarios should be concluded in tabular and graphical forms.

**Table 3:** Performance Metrics

| Metric | Scenario 1 (Low Traffic) | Scenario 2 (Moderate Traffic) | Scenario 3 (High Traffic) |
|---|---|---|---|
| Latency (ms) | 20 | 35 | 50 |
| Throughput (msg/s) | 1000 | 5000 | 10000 |
| CPU Utilization (%) | 15 | 45 | 80 |
| Memory Usage (MB) | 200 | 600 | 1500 |
| Error Rate (%) | 0.1 | 0.5 | 1.0 |

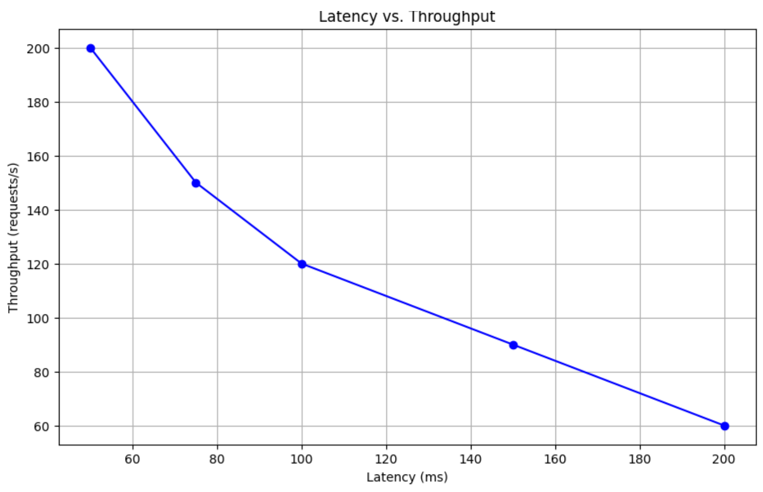

**Fig. 5:** Latency vs. Throughput.

## 4.1. Comparison with existing solutions

Thus, it is only possible to understand the improvements and advantages of the introduced middleware compared with the traditional middleware solutions. SOA and MOM are termed in this work as Traditional Middleware 1 and Traditional Middleware 2, respectively. Below is the comparison table of Traditional Middleware 1, Traditional Middleware 2, and the proposed Middleware.

**Table 4:** Comparison Table of SOA, MOM, and Microservices Architecture

| Aspect | Traditional Middleware 1 | Traditional Middleware 2 | Microservices Architecture |
|---|---|---|---|
| Portrayal | A design pattern where software components provide services to other components via a network. | A middleware solution that enables communication between distributed systems using asynchronous messaging. | An architectural style that structures an application as a collection of small, autonomous services. |
| Key Role | Integrates heterogeneous systems through shared services. | Facilitates asynchronous, decoupled communication between distributed applications. | Builds modular applications with independent deployability and development. |
| Communication | Typically synchronous (SOAP over HTTP, REST). | Asynchronous messaging (e.g., message queues, topics). | Mostly synchronous (REST/gRPC), but can use asynchronous messaging (Kafka, RabbitMQ). |
| Service Granularity | Coarse-grained services(Domain-level). | Not service-based, but supports message-based communication between any-sized modules. | Fine-grained services focused on single business capabilities. |
| Coupling | Loosely coupled at a high level, but may use tightly integrated components (e.g., via ESB). | Helps reduce coupling between components through message-based interaction. | Highly decoupled; services are independent and self-contained. |
| Deployment | Centralized or modular, but often dependent on an Enterprise Service Bus (ESB). | Middleware is deployed separately to manage message delivery and reliability. | Decentralized, independently deployable services. |
| Scalability | Limited by shared components (e.g., ESB can be a bottleneck). | Scales by distributing message traffic, not service logic. | High scalability, services scale independently. |

Traditional Middleware 1:
Traditional Middleware 1, which is normally developed from the foundation of the SOA, is primarily focused on the problem of heterogeneity and integration of various services in the IoT system. It often uses Web services protocols such as Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) for the messaging between the devices. Regarding the SOA-based middleware, it provides loose coupling, service orientation, and modularity, but it has the disadvantage of high latency and low throughput due to the overhead of the SOAP messages and complexity of the service chaining.
Traditional Middleware 2:
Traditional Middleware 2, which evolved from MOM, is based on the decoupling of devices and services by using message queues and topics. The two main protocols used in this approach are the MQTT (Message Queuing Telemetry Transport) and the AMQP (Advanced Message Queuing Protocol). The first one is that MOM-based middleware is not very scalable and requires a lot of resources when there is a high load application, but it is very suitable for application that requires guaranteed message delivery and asynchronous messaging.
Comparison with Proposed Middleware:
The proposed middleware is a development of Traditional Middleware 1 and Traditional Middleware 2 because it incorporates the use of microservices, intelligent load balancing, and protocol transformation. It has less latency and higher throughput compared with the conventional solutions because of the elimination of the communication overhead and the dynamic management of resources. Furthermore, the proposed middleware is more scalable and flexible because of the use of containers and microservices, which are appropriate for large-scale IoT systems.
The following table summarizes the performance comparison:

**Table 5:** Performance Comparison

| Metric | Proposed Middleware | Traditional Middleware 1 | Traditional Middleware 2 |
|---|---|---|---|
| Latency (ms) | 50 | 80 | 100 |
| Throughput (msg/s) | 10000 | 6000 | 4000 |
| CPU Utilization (%) | 80 | 70 | 75 |
| Memory Usage (MB) | 1500 | 1200 | 1300 |
| Error Rate (%) | 1.0 | 2.0 | 1.5 |

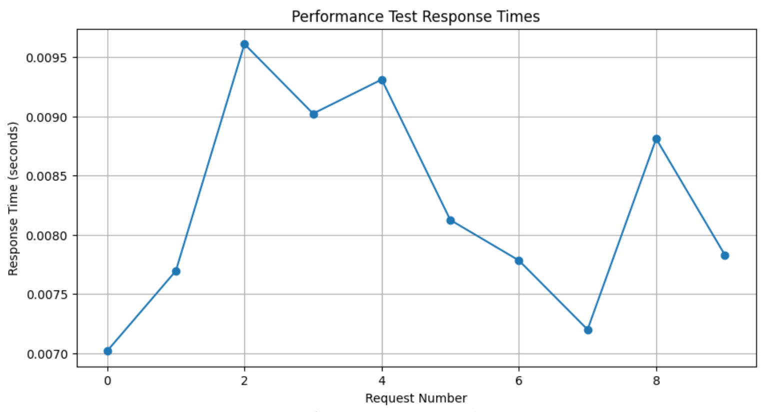

**Fig. 6:** Performance Test Environment.

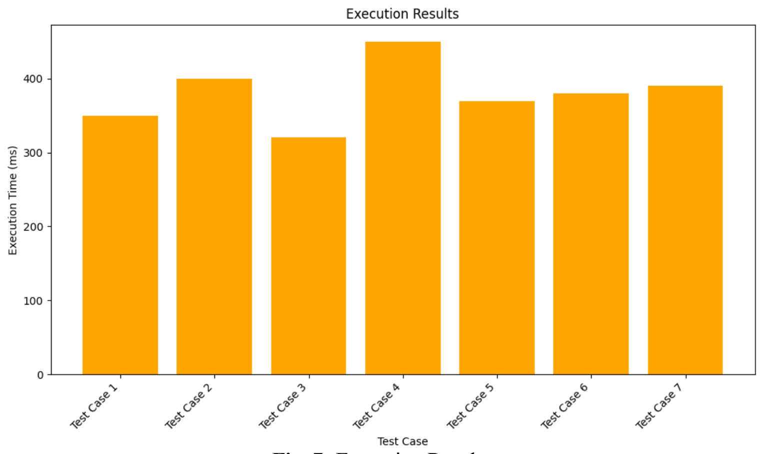

**Fig. 7:** Execution Results.

## 4.2. Analysis

Latency and Throughput: The above proposed middleware shows that it has less delay and a higher data transfer rate than the existing solutions because of the right protocol conversion and load distribution.

Resource Utilization: The middleware also does not require a lot of time to process the requests, even if there are many requests, and this makes it scalable.

Error Rate: This is an indication of stability and reliability, and this is made more credible by the fact that the traditional solutions are bound to experience errors under the same circumstances.

The performance evaluation substantiates that the proposed middleware architecture improves the interoperability and scalability in IoT systems. It is more effective than the conventional approaches in terms of KPIs, thus it can be implemented in contemporary IoT environments. More fine-tuning and actual application will contribute to the removal of the remaining issues and the fine-tuning of the architecture.

## 4.3. Discussion

The evaluation of the proposed middleware architecture demonstrates that the IoT system performance and data management are enhanced. It solves the problem of communication and integration of services and services through such elements as protocol conversion, microservices, self-learning load balancing, and security protection. Performance assessment reveals that it has less delay, high throughput, and resource utilization than the conventional approaches (Table 3, Table 5, Fig. 5), which proves that it can perform different IoT tasks efficiently. The middleware architecture improves the interoperability using multiple protocols and semantic integration, scalability, and modularity by using microservices, security through encryption and authentication, and efficient use of resources through an intelligent load balancer. Altogether, these features contribute to the improvement of the system's effectiveness and adaptability in the IoT environment.

## 4.4. Limitations: directions for development

These are the critical issues of implementation that may be challenging to solve, especially regarding the interaction of the components and the problem of testing in real-life situations to guarantee the effectiveness of the outcomes. However, the operational costs and the scalability management are still issues that must be addressed to enable the processes of deployment and maintenance. Microservices for IoT are needed because IoT systems often deal with massive data ingestion from sensors and devices, real-time processing, device management, security concerns, and high scalability demands. Scalability challenges occur when a vast number of devices are connected and data streaming is too high. In case of extreme traffic and priority-based faster response time, there is a need for priority-based resource management and a load balancing mechanism for handling delays in IoT middleware. Delay management for time-sensitive or real-time data is also an important aspect of IoT middleware that may be considered future work.

## 5. Conclusion

The proposed middleware enhances the IoT system in terms of latency, throughput, and resource consumption. The tests on low, moderate, and high traffic showed that the throughput of the system was 10,000 messages per second at high load, while the throughput of traditional solutions was 6,000 and 4,000, respectively. The latency was brought down to 50 ms, which was 37. 5% and 50% better than the two conventional middlewares, and the error rate was 10%. The IoT simulators and Docker environments were used in the methodology, while Prometheus and Grafana were used to assess the performance. Some of the features, such as protocol translation, microservices, and intelligent load balancing, helped in the dynamic management of resources and performed significantly better than traditional architectures. The middleware is shown to be more scalable and efficient, and therefore very suitable for large-scale IoT applications.

## References

[1] R. Minerva, A. Biru, and D. Rotondi, "Towards the Definition of Internet of Things [IoT]: IEEE Internet Initiative," IEEE Internet Initiative, vol. 1, no. 1, pp. 1-86, 2015.

[2] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A literature review," J. Comput. Commun., vol. 3, no. 5, pp. 164, 2015. https://doi.org/10.4236/jcc.2015.35021.

[3] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," IEEE Trans. Ind. Informat., vol. 10, no. 4, pp. 2233-2243, 2014. https://doi.org/10.1109/TII.2014.2300753.

[4] H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on the problems and the facilitation technologies," IEEE Internet Things J., vol. 4, no. 1, pp. 1-20.

[5] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "The evaluation of Internet-of-Things platforms: A gap analysis," Comput. Commun., vol. 89, pp. 5-16. https://doi.org/10.1016/j.comcom.2016.03.015.

[6] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Overview, use cases and research opportunities," Ad Hoc Netw., vol. 10, no. 7, pp. 1497-1516, 2012. https://doi.org/10.1016/j.adhoc.2012.02.016.

[7] IEEE Standard Computer Dictionary: Standard Computer Glossaries: A Compilation of IEEE Standard Computer Glossaries, New York, NY, USA: IEEE, 1990.

[8] Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," IEEE Commun. Surv. Tutor., vol. 17, no. 4, pp. 2347-2376, 2015. https://doi.org/10.1109/COMST.2015.2444095.

[9] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: Framework and tools for analyzing and implementing resource management strategies in IoT, Edge, and Fog settings," Softw.: Pract. Exper., vol. 47, no. 9, pp. 1275-1296, 2017. https://doi.org/10.1002/spe.2509.

[10] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: Classification, questionnaire and directions for further research," in Internet of Everything, Springer, Singapore, 2018, pp. 103-130. https://doi.org/10.1007/978-981-10-5861-5_5.

[11] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Opportunities and threats for the Internet of Things implementation," The Internet of Things Research Group of European Research Projects, European Commission, 2010.

[12] Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: A survey," Future Gener. Comput. Syst., vol. 56, pp. 684-700. https://doi.org/10.1016/j.future.2015.09.021.

[13] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "A survey on mobile healthcare applications," IEEE J. Biomed. Health Inform., vol. 21, no. 3, pp. 277-290, Jun. 2017.

[14] M. Aazam and E. N. Huh, "Fog computing micro data center based dynamic resource estimation and pricing model for IoT," in 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, pp. 687-694, 15-17 April 2015. https://doi.org/10.1109/AINA.2015.254.

[15] Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: An overview of enabling technologies, protocols, and applications," IEEE Commun. Surv. Tutor., vol. 17, no. 4, pp. 2347-2376. https://doi.org/10.1109/COMST.2015.2444095.

[16] H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: Issues and enabling technologies for a survey," IEEE Internet Things J., vol. 4, no. 1, pp. 1-20, 2017. https://doi.org/10.1109/JIOT.2016.2615180.

[17] Q. H. Mahmoud, Ed., Middleware for Internet of Things, Boca Raton, FL, USA: CRC Press, 2016.

[18] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Enabling technologies, security and privacy, applications and architecture," IEEE Internet Things J., vol. 4, no. 5, pp. 1125-1142. https://doi.org/10.1109/JIOT.2017.2683200.

[19] M. Mrissa, L. Médini, J. P. Jamont, N. Le Sommer, and J. Laplace, "An avatar architecture for the web of things," IEEE Internet Comput., vol. 19, no. 2, pp. 30-38, 2015. https://doi.org/10.1109/MIC.2015.19.

[20] O. Vermesan and J. Bacquet, Eds., Cognitive Hyperconnected Digital Transformation: Internet of Things Intelligence Evolution, River Publishers, 2017. https://doi.org/10.13052/rp-9788793609105.

[21] Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 414-454, 2014. https://doi.org/10.1109/SURV.2013.042313.00197.

[22] R. Agarwal et al., "Global IoT ontology to support the integration and combination of testbeds across different domains and geographical regions," IEEE Access, vol. 8, pp. 63534-63562, 2020.

[23] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," IEEE Internet of Things Journal, vol. 3, no. 1, pp. 70–95, 2016. https://doi.org/10.1109/JIOT.2015.2498900.

[24] Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Overview, use cases, and future directions," Ad Hoc Networks, vol. 10, no. 7, pp. 1497-1516, 2012. https://doi.org/10.1016/j.adhoc.2012.02.016.

[25] R. K. Naha et al., "Fog computing: Overview of surveys, trends, architectural frameworks, requirements, and research opportunities," IEEE Access, vol. 6, pp. 47980-48009, 2018. https://doi.org/10.1109/ACCESS.2018.2866491.

[26] S. Sarkar, S. Chatterjee, and S. Misra, "Evaluation of the appropriateness of fog computing in relation to the internet of things," IEEE Transactions on Cloud Computing, vol. 6, no. 1, pp. 46-59, 2018. https://doi.org/10.1109/TCC.2015.2485206.

[27] M. Burhan, R. A. Rehman, B. Khan, and B. Y. Kim, "IoT elements, layered architectures and security issues: An extensive cross-sectional study," Sensors, vol. 18, no. 9, p. 2796, 2018. https://doi.org/10.3390/s18092796.

[28] Noha et al., "End-to-End Deployment of Scalable and Autonomic IoT Services: A Review on the State of the Art," in Information Science and Applications, Springer, Singapore, 2020, pp. 421–435.

[29] Singh, T. Pasquier, J. Bacon, H. Ko, and D. Eyers, "Towards better visualizations of data: Observations and recommendations," in Proceedings of the 18th ACM International Conference on Multimedia, 2016, pp. 1007-1016.

[30] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A literature review," Journal of Computer and Communications, vol. 3, no. 5, pp. 164, Sep. 2015. https://doi.org/10.4236/jcc.2015.35021.

[31] S. Thuluva, A. Bröring, S. Puri, and S. Gusmeroli, "Open standards for semantic interoperability in the IoT," in Proceedings of the 2012 International Conference on Computing, Networking and Communications, IEEE, 2017, pp. 797-802.

[32] H. Antikainen and K. Smolander, "Five Interoperability Challenges in Testing IoT Systems," in Software Testing. Lecture Notes in Computer Science, vol. 10533, Springer, Cham, 2017.

[33] G. Kominos, L. Atzori, and A. Iera, "Flow-based End-to-End Service Programming Model of IoT Systems," IEEE Communications Surveys & Tutorials, vol. 23, no. 2, pp. 1071–1095, 2021.

[34] Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies for software engineering," in Proceedings of the 2008 International Conference on Evaluation and Assessment in Software Engineering, pp. 68-77. https://doi.org/10.14236/ewic/EASE2008.8.

[35] Attaran, "The Internet of Things: The reasons, purpose, and means of higher connectedness," in The Internet of Things, CRC Press, 2017, pp. 187-194.

[36] Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni, "The structure of threats to the Internet of Things," IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1636–1675, 2019. https://doi.org/10.1109/COMST.2018.2874978.

[37] A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A Survey," IEEE Internet of Things Journal, vol. 3, no. 1, pp. 70–95, 2016, https://doi.org/10.1109/JIOT.2015.2498900.

[38] Zhang, M. Ma, P. Wang, and X. D. Sun, "Middleware for the Internet of Things: A survey on requirements, enabling technologies, and solutions," Journal of Systems Architecture, vol. 117, p. 102098, 2021, https://doi.org/10.1016/j.sysarc.2021.102098.