Comparison of RabbitMQ against Message Brokers to distribute messages across a secure Network Tunnel for Small Business Operations

Aaron Hassan Robinson - u3224074
Norbu Chogyel Tobgyel - u3233707
Mustafa Siddiqui - u3217273

***Abstract*–Message oriented middleware (MOM) is a critical piece of infrastructure for private networks to deliver data to locations across a network. As security becomes a growing concern, private networks separated by a demilitarized zone are becoming commonplace. This research report evaluates performance and security of AMQP across RabbitMQ and compares it with other similar brokers, and protocols, and presents results for both producing and consuming messages across a private network and tunnel.**

Keywords - RabbitMQ, ActiveMQ, throughput, message sizes, AMQP, MQTT

I. INTRODUCTION

Business organizations today use several middleware to fulfill various purposes; to mitigate certain limitations and meet dynamic demands. In telecommunication, Message Oriented Middleware (MOM) are key for building systems that dynamically meet demands. They are used to interconnect independent software systems, allowing for development using various technologies[1].

Message Brokers are MOMs that facilitate communication between several applications. They comprise queues to push messages into, from which the destination servers can consume from. The message queue's ability to scale to carry various loads of data and uninterrupted operation even while failure makes brokers critical for implementation in systems.

However, the implementation of MOMs on top of other middleware (such as proxies, firewall, routers) will leave exposed open ports (although, they are necessary for communication), which comprises security. Using a single network tunnel can potentially mitigate this issue, as this leaves less room for attackers to exploit the network infrastructure.

In this paper, RabbitMQ with AMQP implemented in a network structure (which will be discussed more in the design and methodology section) will be tested via their performance over a single tunnel, and compared to some other message brokers functioning in the same network. The recorded data will be evaluated to see which broker is suitable for implementation in realistic business operations.

RabbitMQ enables efficient communication among several applications, which otherwise, would have been a very complex process involving hgh data traffic. This is done through the provision of a very robust messaging system Being a message broker, it implements AMQP and over time has been extended with plug-ins to support other protocols, i.e., Streaming Text Oriented Messaging Protocol (STOMP) , And Message Queuing Telemetry

Transport (MQTT). RabbitMQ uses the Publish-Subscribe Model, where data is distributed to the intended subscribers. The appropriate publisher and subscriber is selected using topic based concept, and data transmission between the two is looked after by the broker. The producers publish the messages into numerous queues, which are bound with exchange through various techniques, such as Fan Out, Direct, Topic, and Header Exchange [2]. From the queues, the consumers consume the messages.

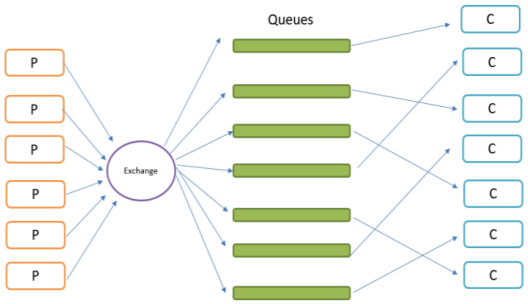

Fig. 1: Architecture of RabbitMQ involving Producers (P) and Consumers(P)

Ivanović et al. [3] emphasize RabbitMQ's ability to handle high message loads with fault tolerance and clustering. In previous research, RabbitMQ has been extensively studied due to its adoption of the Advanced Message Queuing Protocol (AMQP). AMQP provides platform neutrality, allowing it to support multiple programming languages and system configurations, making it highly versatile for distributed applications.

In this specific study, AMQP is implemented in RabbitMQ.. The protocol in general is popular for MOMs, which is widely implemented in communication between enterprises and business organizations. It can be deployed on both cloud services and on physical premises. The protocol can utilize the publish/subscribe model, while maintaining reliability, security, and interoperability[4].

II. LITERATURE REVIEW

A. Protocol analysis and its related works:

There are several messaging protocols that are used in communication, such as HTTP used for REST transportation using the request/response model, while AMQP and MQTT are message-oriented protocols that are based on the publish subscribe model.

MQTT is the standard protocol used in the IoT domain [5], which is inexpensive and lightweight. MQTT is more appropriate for devices requiring less memory and power due to its minimum requirements and miniscule code overload. It also provides reliable communication for unstable networks as it is built on TCP. MQTT uses SSL for security.

AMQP [6] is mainly designed for message-oriented applications. The protocol avoids sending any redundant and unproductive data due to its binary form, and has no limit for message size. It is also built on TCP, and guarantees successful message delivery. It uses two protocols of security (TLS/SSL and SASL), and supports two formats for QoS for better message delivery.

HTTP, till date, remains popular for end-to-end communication, with the daily increase in the number of network appliances. It is a text-based protocol with no size limitations for the headers or payloads, It runs on TCP and uses TLS/SSL for security, but does not provide QoS. It is not used in IoT as much due to it's resource hungry-nature[7]

A paper [8] tested the three protocols in terms of performance, latency and CPU usage for varying message sizes and traffic loads to evaluate their real-time performance, in a Smart City public data set. Their results showed that AMQP and MQTT were four times faster than HTTP, and AMQP was found to have the smallest jitter value, which directly correlates to more stable latencies. Moreover, AMQP outshines both MQTT and HTML in ensuring security, as it implements an additional layer of security, i.e., SASL on top of TLS/SSL for user authentication and validation. Tests have proven

AMQP ideal for implementation in message brokers. This paper will further evaluate the performance of AMQP used across several message brokers and whether it will be able to perform within the security constraints of a small business network.

B. *Analysis of Popular Message Brokers:*

MOM is a vital technology in distributed systems, enabling asynchronous communication between loosely coupled software components. Over time, various MOM systems have emerged, each catering to different use cases and system architectures. Two prominent examples of open-source message brokers are RabbitMQ, ActiveMQ, and Kafka. These brokers facilitate message queuing, which is essential for systems requiring resilience, scalability, and fault tolerance.

Some notable brokers for comparison with RabbitMQ are:

a. *Kafka*

Kafka is one of the message brokers with increasing popularity and has its own active community. It uses the partition concept, where messages are classified into different topics, and each of which is divided into several partitions distributed and stored on different brokers, thereby improving parallelism of the system [9]. Despite it being a binary protocol over TCP, Kafka is popular for its relatively high throughput, as it avoids repeated copy operations, uses batching of data to reduce RPC overhead, and data compression techniques. However, due to its complex nature and reliance on Zookeeper, Kafka requires a lot of optimizations for best performance, which require a significant amount of time and resources [10].

b. *ActiveMQ*

Similarly, ActiveMQ, another well-known message broker, implements the Java Message Service (JMS) standard, providing compatibility primarily for Java-based systems. Research has shown that while ActiveMQ excels in environments where Java applications dominate, it faces challenges when integrating non-Java platforms due to its JMS-focused architecture [9]. However, in terms of broker-to-broker communication, ActiveMQ offers superior support for complex topologies through its broker networks, which can be advantageous for large-scale enterprise applications [10].

C. *Related Work on Comparison of Message Brokers in Terms of Performance:*

The study [9] compared five different popular message queuing systems (RabbitMQ, ActiveMQ, Kafka, and two other message brokers), where Kafka was found to be the best choice for large-scale data collection and analysis, as it displayed the highest throughput among all brokers, This makes it suitable for web activity tracking, streaming, monitoring, etc. However, its latency was found to be higher than that of RabbitMQ and ActiveMQ, making it less suitable for businesses requiring low latency.

The paper by Ivanović et al. [3] discusses RabbitMQ's architecture and compares its performance to other message brokers like Apache Kafka, highlighting RabbitMQ's efficient load balancing and fault tolerance capabilities through clustering, as well as its message persistence mechanisms. RabbitMQ's ability to handle cross-platform message delivery via multiple protocols such as STOMP and MSMQ offers flexibility, making it a robust choice for heterogeneous systems.

Another notable comparison study by Dimova and Marinov [11] highlights the differences in performance between RabbitMQ and ActiveMQ under varying workloads. The research found that RabbitMQ outperforms ActiveMQ in environments where message consumption speed is critical, making it more suitable for applications with high-throughput requirements, such as telemetry systems or GPS tracking services. On the other hand, ActiveMQ showed better performance for transactional operations, which are more common in financial applications like credit card payment processing or banking systems [12].

Despite these performance variations, the choice between RabbitMQ and ActiveMQ often depends on the specific requirements of the system

being implemented. For example, the ability of RabbitMQ to accept connections from different platforms via AMQP is advantageous in scenarios requiring cross-platform communication. ActiveMQ, when originally integrated with Java, makes it ideal for enterprise-level Java ecosystems. However, possibilities of running ActiveMQ using a python library will be worth exploring. Additionally, ActiveMQ's performance scalability through broker networks makes it a suitable option for medical systems that manage large volumes of data [13].

In summary, different message brokers have their own strengths; the selection of a message broker largely depends on system-specific factors such as the platform ecosystem, message throughput, and cross-platform communication needs. This paper aims to build on the existing research by providing a focused performance evaluation of RabbitMQ; specifically in distributed small business environments, particularly emphasizing its use with AMQP to securely distribute messages across a network.

III. DESIGN AND METHODOLOGY

To test and verify the performance of message brokers, a prototype/representative of the network infrastructure of a small business network was created, which involves a client connecting to an internal web server, which is made accessible through the setup of a proxy through a firewall. As both brokers use the AMQP (version 1.0) protocol, which enables transmission of messages through TCP/IP connections, a TCP proxy was set up as well to allow messages to pass. This was achieved by using the tools NGINX (version 1.24.0) [14], and Squid (version 6.6) [15]. A network tunnel was implemented using OPENVPN (version 2.6.3) [16], which implements the OSI layer 2 - 3 secure network extension using the SSL/TLS protocol. To prevent any external traffic from interfering in the process, a Demilitarized zone (DMZ) was implemented between the proxy and the VPN router using Firewalld.

For the overall implementation of the design, VMware was used to set up several Ubuntu virtual machines to run local environments.

The VPN router, proxy server, and web server all were configured with 2 cores each, 4GB RAM, 20GB disk storage running Ubuntu Server 24.04.

Some notes about the above design are:

1. The FRONTEND is the default shared network in VMWare, which is accessible through the host computer.
2. The DMZ and BACKEND networks are the custom subnets set up using the settings of VMWare.
3. The three VMs were each assigned an interface from the respective subnet.
4. Each link not part of the VPN were assigned a static IP address through the netplan cloud configuration files
5. The proxy and web server were temporarily given FRONTEND interfaces for installation of necessary packages and connection to the internet.

The server of the message broker (i.e., the web server in the model) displays the management web dashboard, which displays live data about transmission of messages to and fro the server. To test the performance of each broker, the producer-consumer model is used while using the AMQP protocol.

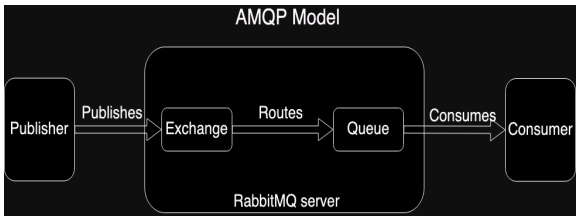

Fig. 2. AMQP model involving the producer, consumer, and RabbitMQ

Some key aspects of the model are:

A. *Exchanges*

The publisher sends messages to the exchanges in the RabbitMQ (version 4.0.2) server, which in turn publishes the data to queues using bindings (the relationship between an exchange and a queue). The chosen exchange type for this scenario is direct exchange, which is based on the message routing key. This is ideal for unicast routing, although multicast routing can utilize this a well. In the settings of the exchange, the durability was set to True, which will allow it to survive broker restarts.

B. Routes

In direct exchanges like this, messages are delivered to queues using a specific routing key. For instance, with a routing key k:

a. A queue binds to an exchange with a routing key k
b. A message with key k arrives at the exchange, which in turn will be routed to an appropriate queue with key k.
c. If multiple queues are bound to the exchange with the same key k, the exchange will route the message to all queues bearing the key k.

C. Queues

They act as temporary buffers that temporarily store messages, which are soon to be consumed by the applications.

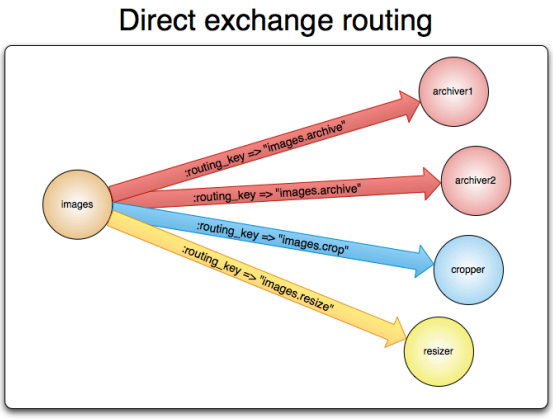

Fig. 3. Visual representation of direct exchange routing involving routing keys

The Producer/Consumer will be the basis for testing the performance of RabbitMQ and Mosquitto, where messages of varying sizes and numbers will be pushed one by one to designated queues for the consumers to consume messages from. This will result in varying rates of throughput, which will be compared and analyzed. Theoretically, the throughput rate of both the producer and consumer will decrease with increase in size of the messages.

TABLE I. SUMMARY OF TYPE OF VARIABLES

Independent variables:	• Type of Message Broker Used (RabbitMQ vs ActiveMQ) • Packet Size (256 bits - 8 MB) • Number of Messages (1000-10, depending on the packet size)
Dependent variables:	• Throughput • Response time • Rate of Publish/Consumption
Controlled variables (Constants):	• Network Topology ◦ The messaging protocol implemented (AMQP vs MQTT) • Language consumer / producer model created in (Python) • Library used for benchmarking (timeit) • Virtual Machine Specifications • Message Type (Text/Plain)

Limitations of the model to consider are:

1. The model only tests production and consumption throughput of one client/one destination server, which does not translate well in real-world applications.
2. Performance limitations in the model allows for capturing of only a single stream of network, with limited resources (running the network in a single computer). Ideally you want to test the performance across a range of queues / topics.
3. We could test potential numerous devices in DMZ and backend, and measure through

Additional Design Challenges

Different message brokers were swapped out on the web server device. This was to demonstrate the furthest point in the network.

ActiveMQ has no native supporting Python libraries. Attempts were made to use the python AMQP library, however the version of ActiveMQ installed (6.1.3) only supported AMQP V1 messages, where the python AMQP library can only generate messages using AMQP 0.9.1. For this reason we used the QPID Proton library.

Randomizing data

All data sent across the network was randomized to avoid any caching effect from the proxy server used in the network.

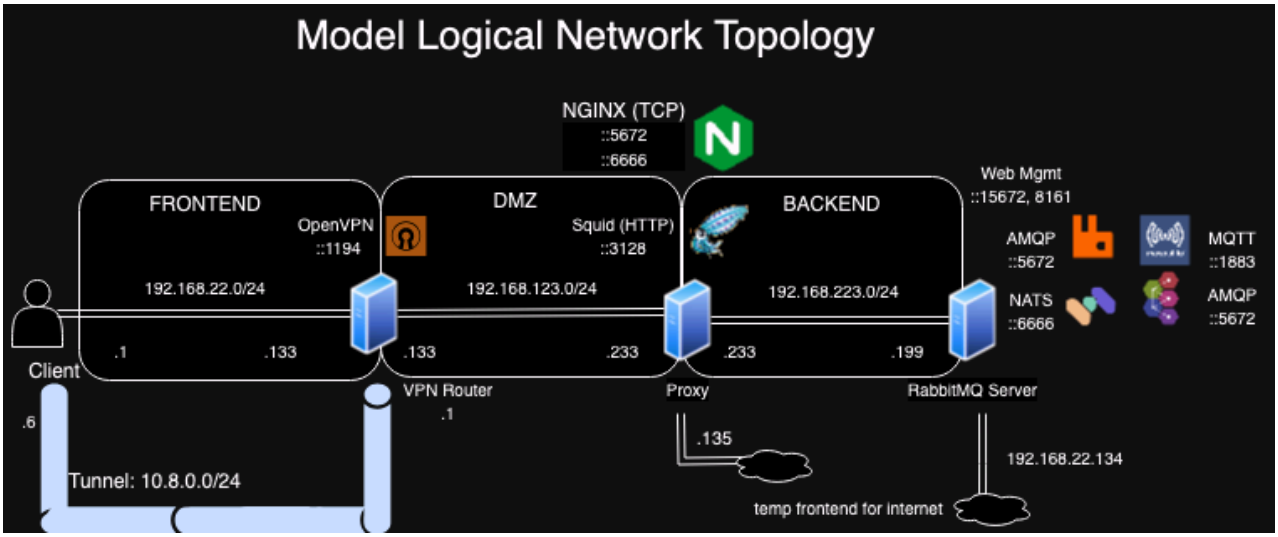

Fig. 4. Structural Design of the proposed implementation to test performance analysis

IV. RESULTS

For better visualization of the throughputs obtained from varying message sizes, the following graphs were plotted for comparison:

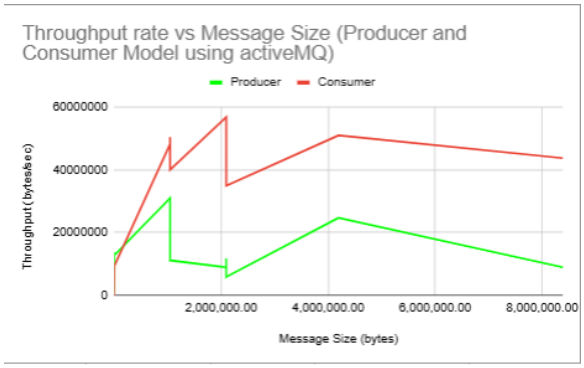

Fig. 5. Message Size vs Throughput rate in RabbitMQ using the Producer/Consumer Model

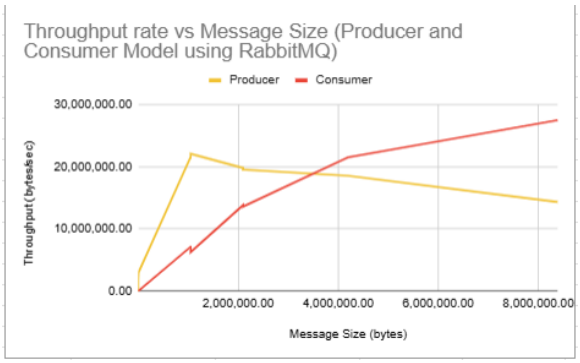

Fig. 6. Message Size vs Throughput rate in ActiveMQ using the Producer/Consumer Model

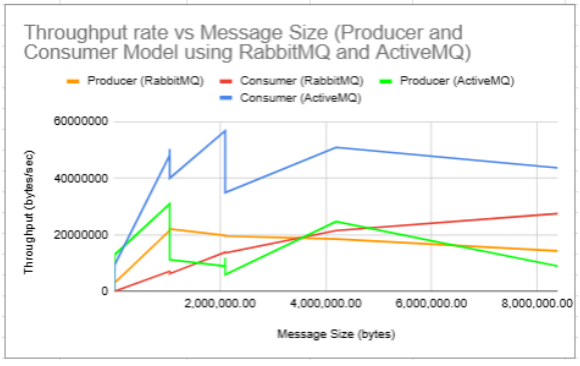

Fig. 7. Message Size vs Throughput rate in RabbitMQ and ActiveMQ using the Producer/Consumer Model

A. Testing of Miscellaneous Brokers and its associated protocols:

a. Memphis

Memphis is a message broker that uses the NATS protocol. At the time of writing this report, AMQP and MQTT have not come to Memphis. There were issues with the software that was not able to be overcome, one of the main issues was that extending the default message size from 1MB was not possible due to a bug in the web console. From our data we found that at small message sizes (256b) Memphis had a throughput rate roughly double RabbitMQ: 5,669B/S vs 2,830B/S.

b. Mosquitto

Mosquitto was tested using the MQTT protocol. Our group wanted to test the performance of MQTT with additional security attributes of TLS and certificates to see how it's performance compared against AMQP, however due to time constraints, we only completed a test without TLS or certificates, and no persistent storage. The throughput achieved from this was magnitudes larger than any broker using AMQP, however since there were no messages being written to memory and virtually no security, it meant that a client had to be listening to receive messages, or they would be lost, and the security risk associated with this protocol was immense.

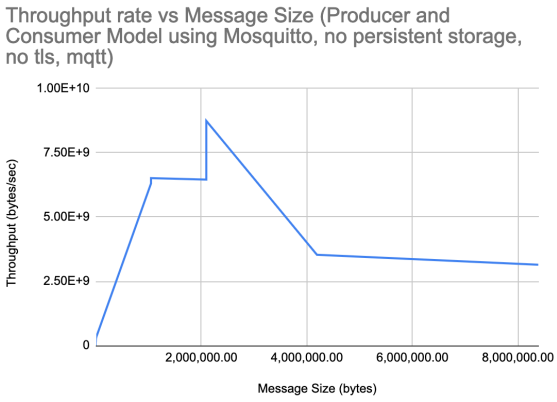

Fig. 8. Message Size vs Throughput rate in Mosquito using the Producer/Consumer Model, displaying inconsistent results due to absence of security and persistent storage

TABLE II: COMPARISON OF PEAK THROUGHPUT RATES OF RABBITMQ AND ACTIVEMQ USING THE PRODUCER/CONSUMER MODEL

Broker	RabbitMQ (AMQP)		ActiveMQ (AMQP)	
	Producer	Consumer	Producer	Consumer
Peak through-put:	22,059 Kb/sec	27,475 Kb/sec	30,972 Kb/sec	56,805 Kb/sec
Message Size and Number of Messages	1049 kB, 100 units of Messages	8389 kB, 10 units of Messages	1049 kB, 100 units of Messages	2097 kB, 100 units of Messages

V. DISCUSSION

It was observed that ActiveMQ displayed the highest levels of throughput using the AMQP protocol across a single queue, in terms of producing and consuming. Although it had a greater overall throughput, it was also noted that there was a greater inconsistency in results from ActiveMQ when compared with RabbitMQ.

Both ActiveMQ and RabbitMQ achieved the greatest throughput with payload message sizes of 1MB. It can be observed a drop off takes place as the message size increases beyond this.

Areas of improvement:

Additional testing that would have been valuable to complete would be to stress test a range of queues as well as payload sizes. Along with this, we completed all tests along the furthest point of the network, from the client to the internal server. Additional research to further demonstrate performance would prove valuable in locations inside the DMZ, and backend as well as further testing into different protocols. Security research into AMQP vs MQTT and the trade off between throughput performance and security could also be conducted and analyzed to see which protocol performs better. Mosquitto with no persistence, no TLS or certificates had a peak throughput of 8.7297 GB/s for message payloads that were 2MB large. This could potentially prove valuable for IoT or low risk devices inside an internal network. Due to time constraints, we could not test the QPID Python library against RabbitMQ which may have provided some more valuable insight and accuracy in results. It would also be worth testing the latency of each broker to further check their reliability in business organizations.

VI. CONCLUSION

An experiment to test the performance of RabbitMQ using AMQP against ActiveMQ and other brokers implemented in a small business network was conducted. It was found that peak performance of both brokers was achieved at the message size of 1mb, where ActiveMQ demonstrated higher throughput (specifically in the consumer model) than RabbitMQ, despite more inconsistencies than RabbitMQ. It is concluded that each broker is situationally superior; RabbitMQ can be used in online services due to lower latencies, while applications requiring raw throughput can implement ActiveMQ. Attempts were made to test Memphis and Mosquitto, but consistent results were not achieved due to time constraints and bugs.

In future work, it would be helpful to test various ranges of queues and payload sizes, and degree of security of various protocols, to further optimize the performance of business networks.

REFERENCES

[1] V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER), Craiova, Romania, 2015, pp. 132-137, doi: 10.1109/RoEduNet.2015.7311982.

[2] A. Pathak and C. Kalaiarasan, "RabbitMQ Queuing Mechanism of Publish Subscribe model for better Throughput and Response," 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT), Erode, India, 2021, pp. 1-7, doi: 10.1109/ICECCT52121.2021.9616722.

[3] Ivanović, S., Dedić, G., & Avdaković, S., "Performance evaluation of RabbitMQ message broker in distributed systems," International Conference on Information and Communication Technology, 2018.

[4] N. Basavaraju, N. Alexander and J. Seitz, "Performance Evaluation of Advanced Message Queuing Protocol (AMQP): An Empirical Analysis of AMQP Online Message Brokers," 2021 International Symposium on Networks, Computers and Communications (ISNCC), Dubai, United Arab Emirates, 2021, pp. 1-8, doi: 10.1109/ISNCC52172.2021.9615705.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” IEEE communications surveys & tutorials, vol. 17, no. 4, pp. 2347–2376, 2015

[6] O. Standard, “Oasis advanced message queuing protocol (amqp) version 1.0,” International Journal of Aerospace Engineering Hindawi www. hindawi. com, vol. 2018, 2012.

[7] I. Grigorik, “Making the web faster with http 2.0,” Communications of the ACM, vol. 56, no. 12, pp. 42–49, 2013.

[8] C. B. Gemirter, Ç. Şenturca and Ş. Baydere, "A Comparative Evaluation of AMQP, MQTT and HTTP Protocols Using Real-Time Public Smart City Data," 2021 6th International Conference on Computer Science and Engineering (UBMK), Ankara, Turkey, 2021, pp. 542-547, doi: 10.1109/UBMK52708.2021.9559032.

[9] G. Fu, Y. Zhang and G. Yu, "A Fair Comparison of Message Queuing Systems," in IEEE Access, vol. 9, pp. 421-432, 2021, doi: 10.1109/ACCESS.2020.3046503.

[10] S. Das, S. Goswami and S. C N, "Optimizing Apache Kafka Deployments - Configuration customization is all you need," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-9, doi: 10.1109/ICCCNT56998.2023.10307242.

[11] A. Dimova and N. Marinov, "Comparative Performance Evaluation of RabbitMQ and ActiveMQ," *Proceedings of the 12th International Conference on Middleware*, 2017.

[12] T. Fu, and Y. Wei, "Transactional messaging in financial applications: RabbitMQ vs ActiveMQ," *Journal of Financial Technology*, vol. 4, no. 3, pp. 73–84, 2019.

[13] H. Zhang, L. Xu, and J. Zhang, "Middleware technologies in medical record management," *Health Informatics Journal*, vol. 21, no. 2, pp. 134-145, 2017.

[14] “nginx,” nginx.org. https://nginx.org/en/

[15] “squid : Optimising Web Delivery,” www.squid-cache.org. https://www.squid-cache.org/

[16] OpenVPN, “OpenVPN,” OpenVPN, 2018. https://openvpn