

An Endogenous and Self-organizing Approach for the Federation of Autonomous MQTT Brokers

Marco Aurélio Spohn^a

Federal University of Fronteira Sul, Chapecó, SC, Brazil

Keywords: Publish/Subscribe, Broker Federation, MQTT, Internet of Things, Self Organizing Protocols.

Abstract: Many applications for the Internet of Things (IoT) use the publish/subscribe (P/S) communication paradigm. Among the most representative protocols, there is MQTT. Its basic architecture relies on a single server/broker: publishers send data topics to the broker, and then it forwards the data to subscribers. Having a single server may make things easier for configuration and management; however, there is room for a potential bottleneck, besides being a single point of failure. Clustering servers usually address scalability in MQTT broker deployment; however, most solutions are proprietary. Meanwhile, autonomous brokers could be federated together to scale and increase availability. A self-organizing federation proposal already available in the literature implies substantial changes to brokers' inner implementation; furthermore, there has not been any implementation yet. This work explores an endogenous federation approach: design a supporting agent (called federator) that realizes the brokers' federation based on the native P/S mechanism. There are no implied modifications to regular/standard brokers, but it requires changes to the client-side (i.e., publishers and subscribers). This work presents a primary architecture and an initial case study to grasp some fundamentals and benefits of adopting the proposed solution.

1 INTRODUCTION

The Publish/Subscribe (P/S) approach is widely adopted by many Internet of Things (IoT) platforms (Al-Fuqaha et al., 2015), being Message Queuing Telemetry Transport (MQTT) (Standard, 2020) one of the prominent representatives among the P/S protocols. In MQTT, there is a server (broker) responsible for managing the relationship between subscribers (i.e., those willing to receive specific information) and publishers (i.e., those eventually making the information available). It is a consumer/producer relationship: publishers send information/messages to the broker; meanwhile, it delivers the data to all corresponding subscribers.


Developing IoT applications requires efficient communication capabilities, mainly because most end nodes have low computing power, storage capacity, communication, and power source (usually battery powered). Nodes might not be active all the time, making asynchronous communication not just a desirable feature but an essential requirement. Communication between IoT entities and the broker is

paramount for P/S based applications; otherwise, it might impact the supported service's availability.

Even though one expects data topics resulting in small and not so frequent data packets, the broker might be a bottleneck for the overall system performance, besides being a single point of failure. It is possible to resort to more than one broker, allowing us to have some degree of fault tolerance. However, one should expect to have some management issues not present in a single broker configuration: brokers work together so that publications can reach their intended subscribers regardless of their direct connecting broker.

The main advantages of having a federated group of MQTT brokers are:

- There is no single point of failure: clients can choose to associate with any federated brokers.
- Load balancing: whenever there is a possibility to choose among a set of available brokers, and getting what one needs no matter from which broker, it allows adopting mechanisms for load balancing.
- Exploring virtualized topologies or networks' capabilities: full virtualized deployment is realizable by having brokers instantiated in virtual ma-

^a  <https://orcid.org/0000-0002-9265-9421>

chines or containers. Simultaneously, Software Define Networking (SDN) and Network Function Virtualization (NFV) create an environment for endless network topologies. Therefore, the meshes' redundancy capabilities are quite manageable.

Spohn (Spohn, 2020) proposed the first self-organizing solution for the federation of autonomous P/S brokers, providing the primary mechanisms for bringing the brokers together and allowing the proper routing of published topics to their intended targets regardless of the subscribers' brokers. Brokers are assumed to be connected through an overlay/virtual network, allowing any possibility in the network topology. Brokers ignite the federation process with subscribers by building meshes connecting all the brokers with subscribers for the same topic. The solution allows setting the mesh degree redundancy, resulting in a mesh as complex as the underlying network topology allows. As a result, mesh member brokers might be connected through redundant paths, creating the conditions for an enhanced degree of fault tolerance, besides the intrinsic load balance possibilities arisen from having several available brokers.

Nevertheless, no real implementation of the proposed solution is available yet, but the author assumes a protocol realization requires changes to standard brokers' inner implementation. Under other conditions, one could investigate the possibility of realizing the federation at the application layer based on the P/S mechanism itself, without requiring any brokers' changes. In this context, the federator would play more of a supporting role in the whole process.

This work endeavors to provide the first proposal to implement MQTT brokers' federation at an application level/layer. The remainder of this paper has the following structure. Section 2 presents some representative related work. Section 3 summarizes the federation of MQTT brokers used as the basis for this work. Section 4 introduces the proposed endogenous (i.e., P/S based) federation architecture, while Section 5 shows some of its formal properties. In Section 6, a case study is presented as a first realization of the proposed architecture, while Section 7 addresses an initial evaluation scenario. Conclusions and future works are in Section 8.

2 RELATED WORK

Clustering servers usually address scalability in MQTT broker deployment; however, most solutions are proprietary. HiveMQ™ (HiveMQ, 2020) defines an MQTT broker cluster as a distributed system act-

ing as a single logical MQTT broker, allowing clients to interact with the system no matter how many real brokers are active. SwiftMQ™ (SwiftMQ, 2020) builds around the concept of Federated Router Network, where each broker is a SwiftMQ router running a routing engine. Even though the routing process is dynamic, the solution is not self-organized.

The first self-organizing solution (Spohn, 2020) for the federation of autonomous brokers inherits its properties from a multicast protocol designed for Mobile Ad Hoc Networks (MANETs). The basic idea is to have brokers with local subscribers responsible for building and maintaining a mesh structure around themselves. Control overhead is kept to a minimum, having just two control packets: core announcement and mesh membership announcement. Mesh redundancy is configurable, paving the way for a degree of path fault tolerance as long as supported by the underlying topology. Brokers connect through an overlay network, while publishers and subscribers behave as usual (i.e., the brokers' implementation adheres to the federation protocol). Therefore, a new variant/standard for MQTT is required. On the other hand, this work focus on designing an approach that does not require any immediate changes to the MQTT standard. However, the trade-off is on laying the burden on clients (i.e., publishers and subscribers) required to be aware and adhere to the federation protocol by themselves.

Intel proposed Distributed Publish & Subscribe for the Internet of Things (DPS) (Corp., 2020) as a distributed implementation of the P/S communication mechanism. DPS builds a mesh network around subscribers, providing the routing from publishers to subscribers based on subscription topics, with IP multicast support in local subnets. However, DPS does not provide means for the federation of autonomous brokers (i.e., mesh deployment centers on publishers and subscribers, without multiple independent brokers).

Park *et al.* (Park *et al.*, 2018) proposed Direct Multicast-MQTT (DM-MQTT) as a way to address MQTT single broker limitations. As the number of clients connected to the broker increases, there is a direct impact on the communication delay between publishers and subscribers. To reduce communication delays, DM-MQTT employs Software Defined Network (SDN) multicast trees to connect publishers to subscribers, bypassing the centralized broker. Similar to DPS, DM-MQTT is not a federation of autonomous brokers either.

3 FEDERATION OF MQTT BROKERS

This section presents a brief description of the federation mechanism presented in (Spohn, 2020). The proposed approach assumes that a set of autonomous brokers connect through an overlay network. Depending on the underlying topology, it is possible to explore path redundancy in the resulting mesh structures built from the federation process.

The fundamental concept is to have subscribers responsible for organizing the communication subsystem required for the proper routing of publications to every broker with local subscribers. Brokers with local subscribers start by advertising themselves as the core for a new mesh related to the corresponding topic. Core announcements are broadcast to the whole network, allowing any broker to learn who they are and how to get to any existing core. There might be more than one core for the same mesh for a short period, but eventually, all brokers converge to choosing the core with the smallest ID. The remaining core has to announce itself periodically, allowing brokers to keep always the most current information about the core.

To complete the mesh building process, brokers with local subscribers must let their parents know their status by sending a mesh membership announcement (always related to the freshest core announcement). The number of parents depends on the desired mesh degree redundancy. A mesh member is any broker with local subscribers or one connecting children to the core. This way, a mesh membership announcement can cascade back to the core.

With a minimum control overhead, all brokers learn about any existing mesh through the corresponding core announcements. However, a broker only is a mesh member if the broker is required. In their turn, publishers do not need to take part in any mesh: their host broker learns how to get to any mesh (guided by the corresponding core) and send publications towards the core. Once reaching the mesh, the publication needs to be spread all along with the mesh.

The proposed federation protocol assumes changes to the code of the MQTT broker. Therefore, a new variant/standard must arise to provide all the brokers' proper federation mechanisms. In addition to that, there are several open issues not addressed in the initial work. There is not a solution for the relationship between meshes and sets of topics. The virtual topology directly impacts the overall performance, requiring a flexible and efficient solution for managing the topology.

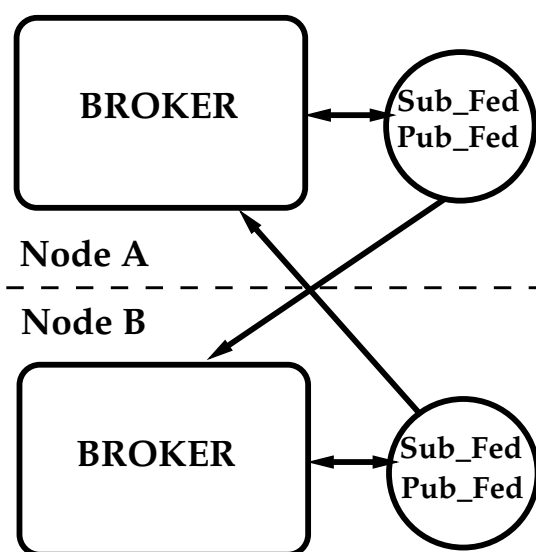


Figure 1: P/S based federation architecture.

4 P/S BASED ARCHITECTURE

In this work, the proposed architecture builds around the P/S subsystem without any required changes to standard MQTT brokers (see Figure 1). The creation and management of meshes depend on gathering the information needed to build them. The dissemination of core and mesh membership announcements proceed through publications and subscriptions. Routing of regular data messages/topics happens similarly.

There are basic application-level processes associated with every broker to perform the roles needed to the federation and routing (i.e., control and data planes):

- **Pub_Fed:** responsible for the publication of control messages (i.e., core and mesh announcements) and regular routing. There is a direct connection to every neighboring broker for the control data plane (i.e., control announcements). Data forwarding will depend on neighboring mesh membership status.
- **Sub_Fed:** responsible for handling control and data packets sent from neighboring brokers. There should be a single instance of such a process associated with any federated broker.

The core federation protocol performs through the proper coordination of the **Pub_Fed** and **Sub_Fed** elements. Core announcements are periodically started from cores (management and regular cores), spreading through the entire virtual topology; that is, a broker must handle core announcements no matter the broker's mesh status. This way, brokers learn how to get to any existing core. A broker only takes part

in the corresponding mesh when the broker has local subscribers or the broker connects (i.e., is a parent to) at least one neighbor to the core (learned from a mesh membership announcement sent from such neighbor).

Cores periodically send new core announcements, uniquely identified by their core ID and sequence number. When brokers receive a new core announcement, they must forward it to its neighboring brokers (other than the sending broker). Any broker learns which neighboring brokers lead to the corresponding core, having as many parents as the allowed mesh redundancy. Once a core announcement reaches a broker, which is also a mesh member, the broker sends a mesh membership announcement to its parents. Such an action occurs for every new core announcement (i.e., mesh membership announcement carries the information regarding the corresponding core ID and sequence number). Brokers with local subscribers, or connecting neighboring brokers towards the core, must send a mesh membership to their parents to properly connect all receiving brokers (i.e., brokers with local subscribers).

Once the core starts a new mesh, it takes some time to connect all brokers depending on how far apart receiving brokers are from each other. Once a mesh membership announcement reaches a non-mesh member parent for the first time, the parent's status transitions to mesh member (i.e., it is an interconnecting broker). The receiving broker must then itself send a mesh membership announcement to its parents. The process might go on until it reaches the core or a mesh member broker that has already sent out the corresponding mesh membership announcement once it received the core announcement previously. A mesh membership announcement also acts as a beacon to track the mesh children's current status. This approach guarantees that every mesh member receives one mesh membership announcement from each child for every new core announcement.

Mesh membership publications happen asynchronously; that is, when a neighbor gets recognized as a parent for a given mesh, a new mesh membership can get published to the corresponding neighbor. The federator must keep control of any recently published mesh membership, each associated with a unique core announcement (i.e., code ID and sequence number).

Regular data packets (i.e., topics' publications) progress toward the mesh by directing the packet to the core. If the source broker is not in the mesh, the packet can reach any particular mesh member other than the core during the routing process. Regardless, once a mesh broker receives the packet, it just needs to be spread over the remaining mesh members: the packet advances to the remaining mesh neighbors

(i.e., parents and children) other than the sending broker. To avoid looping (e.g., when the mesh is a graph with cycles), a data log must be employed to keep records of the most recently forwarded data packets. The size of the log and the minimum time for keeping the entries should be left as a configuration parameter, assuming that it depends on the characteristics (e.g., topology) of the network and the meshes themselves.

4.1 Applications

The proposed architecture's main advantage is that it does not require any changes to standard MQTT brokers. While the burden lies on the application side, it provides a certain degree of freedom for deploying and managing in many ways the whole federation infrastructure.

With the potential virtualization of any computing and communication resources (e.g., virtual machines, containers, software-defined networks, network function virtualization), the federation itself could result as a service. Brokers could run in independent VMs/containers deployed anywhere in a cloud infrastructure. The virtual topology could perform as required by the application, creating the environment fitted to allow the degree of availability and fault tolerance urged by the provided service.

Having the federation protocol incorporated into the broker itself would likely render better overall performance, but employing dedicated containerized brokers associated with every single federator would be a strategy one could leverage to get improved performance results.

5 FORMAL PROPERTIES

This section presents some formal properties related to the safety and liveness of the protocol.

Theorem 1. *For a virtual network topology with n brokers and l links, core announcement overhead is bounded to $O(l)$ publications.*

Proof. Core announcements appear as publications to neighboring brokers in the virtual network. The announcements have unique identifiers (i.e., core ID and sequence number), avoiding looping. An announcement traverses the same virtual link at most twice, happening when links connect pairs of brokers at the same distance to the originating core. Therefore, the total number of publications resulting from every new core announcement is at most $2 \times l$, which is of order $O(l)$. \square

Theorem 2. For a virtual network topology with n brokers and l links, the mesh structure converges after a finite period.

Proof. The protocol converges with no possibility of deadlocks. Theorem 1 proves that the core announcement overhead is bounded by $O(l)$ publications, each one bounded by a well-defined publication delay. From that, we can assume that all brokers, including those with subscribers, get to know the shortest distance to the core and the corresponding neighbors leading to the core (i.e., parent nodes). Brokers with local subscribers must send a mesh membership publication to all parents (bounded to the mesh redundancy), which are known to be closer to the core. Interconnecting brokers become mesh members, allowing all the required mesh membership publications to be published towards the core, accomplishing the mesh construction process. By definition, mesh membership publications are strictly related to unique core announcements. Given that mesh membership is published only once to the broker’s parents, there is no possibility of deadlock. □

6 CASE STUDY

A case study for the proposed solution was designed based on the asynchronous P/S libraries provided by the Eclipse Paho project (Foundation, 2020b). An instance of the federation application handles the interaction with each federated broker, henceforward named the federator (see Figure 2). We take some assumptions to provide a first working environment with essential tweaks for usability and performance analysis:

- The virtual topology is static and known in advance. There are many solutions for building virtual peer-to-peer (P2P) topologies adaptable to a real implementation. For the sake of simplicity, we use a grid topology for exploring path redundancy among brokers.
- There is a set of control topics in use for the proper federation of brokers. Sub_Fed is responsible for subscribing to the following topics:
 - **CORE_ANN:** For receiving core announcements from neighbors, the payload includes information regarding the core ID, sequence number, distance to the core, and the sender’s ID (i.e., the neighbor’s ID sending the core announcement).
 - **MESH_MEMB_ANN:** Carries a mesh membership announcement; payload includes the

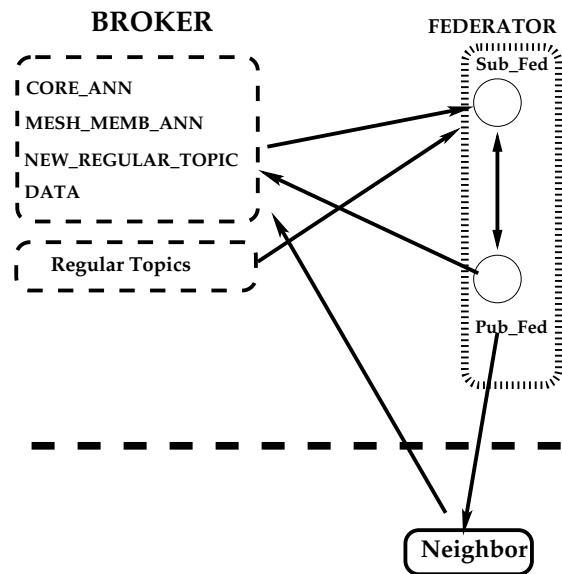


Figure 2: Interactions between federation elements.

corresponding core ID, sequence number (i.e., linked to the corresponding core announcement), and the sender’s ID.

- **NEW_REGULAR_TOPIC:** Any new regular topic subscription, or first publication, must be followed by a corresponding publication of topic details so that the proper mesh building process occurs, including the routing of topic/data packets over the mesh structure.
- **DATA:** Encapsulates any other regular topic. Once reaching a broker with local subscribers, the payload contains a full regular topic packet, which can then be published to reach the proper local subscribers.

There are no modifications required to MQTT brokers; however, as expected, there is some price to pay for that: in order to learn about the related topics which are required to be handled by the federators, any regular subscription, or first publication, have to be informed to the federator (i.e., by publishing the required information through the NEW_REGULAR_TOPIC). Applications must comply with this requirement; otherwise, there is no way to guarantee that publications will reach subscribers associated with different brokers. Therefore, all federators are required to subscribe to regular federated topics (learned about from CORE_ANN and NEW_REGULAR_TOPIC publications) to properly handle the routing of regular topics to the required neighboring brokers, which must receive/relay the publication.

Given that there are some changes needed for regular applications to work in a federated environment,

one could say that clients could have all their regular topic messages encapsulated into DATA topics. Nevertheless, we handle publications the way they happen regularly. Besides that, the number of expected messages for an average application usually outgrows the initial control overhead (i.e., a single publication to the NEW_REGULAR_TOPIC is required). This way, publications are carried by DATA topic messages only when transferred between federators.

A publisher may be associated with any broker wherever there is a supporting instance of the federator. Once a federator receives a regular topic publication, the federator encapsulates the publication into a DATA topic publication, processing it the following way:

- If the federator is not a mesh member for the corresponding topic, the federator relays the packet to one of its parents towards the core.
- If the federator is in the mesh, the federator sends a publication to every mesh neighboring broker (i.e., parents and children) other than the sending broker. That is, the publication spreads only along with the mesh.

7 EVALUATION SCENARIO

One expects performance enhancements when resorting to multiple brokers. However, there is always additional control overhead when such brokers perform together. In this context, the proposed solution's first evaluation scenario targets some of the potential performance benefits. An analysis of fault tolerance and path redundancy capabilities would require extensions to the protocol, leaving for future work.

For the evaluation, we have designed the following scenario:

- A virtual grid topology with nine nodes: a 3×3 grid (see Figure 3).
- Each virtual node is an instance of a Docker container (Docker, 2020) running a Mosquitto MQTT broker (Foundation, 2020a).
- Mesh redundancy has value two, and subscribers execute at node two and node seven. This way, a mesh with path redundancy is established having node two as the core: even though there might be more than one core for a while, eventually, the node with the smallest id (i.e., node two) exerts the core role.

The communication among mesh members' internal elements is depicted in Figure 4, while Figure 5 shows

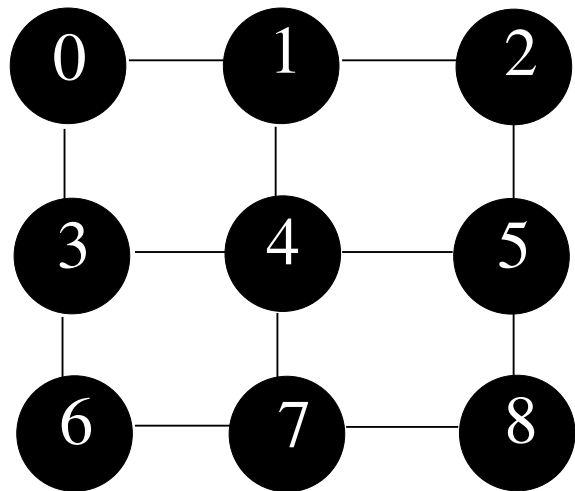


Figure 3: Virtual topology for the evaluation scenario (each node is a Docker container running a Mosquitto broker).

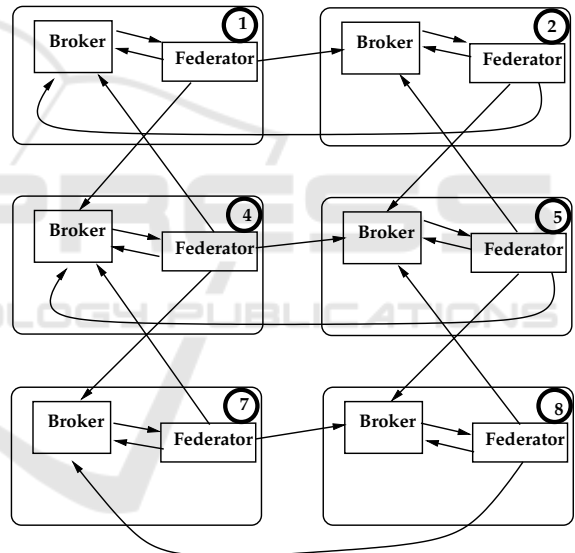


Figure 4: Resulting mesh: communication among mesh members' internal elements.

the parent and children designation. The resulting mesh includes connecting nodes 1, 4, 5, and 8.

A publisher was instantiated at node six (i.e., one hop to node seven and four hops to node two) to explore the shortest and farthest distance to a mesh member with a subscriber. Publications have 64 bytes each, and the publishers send them with an inter-arrival time between 50 and 100 ms (i.e., an average between 10 and 20 publications/s). We use two publication settings: one with 500 publications and the other with 1000 publications. As the primary performance metric, we evaluate the average delay for actually getting the message delivered to each subscriber

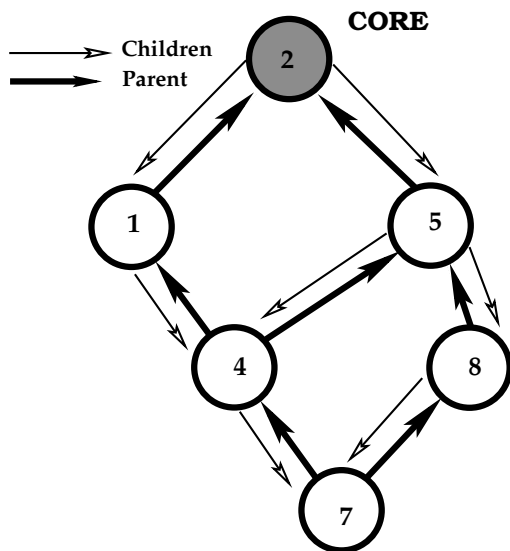


Figure 5: Resulting mesh: parent and children designation.

Table 1: Major parameters for the evaluation scenario.

Parameter	Description	Value/Range
Link delay	virtual link delay	5 ms
Publications' periodicity	inter arrival time for publications	[50..100] ms
Topic payload	number of bytes as topic's payload	64
Mesh redundancy	number of parents for mesh members	2

(Table 1 summarizes the main configuration parameters).

As expected, results for the federated setting (see Table 2) show that the delay experienced by the subscriber at node 7 (i.e., the one closer to the publisher) is almost half the amount registered at node 2. Figure 6 depicts the routing process, showing that redundant transmissions can happen depending on race/timing conditions (e.g., it could take place between brokers 4 and 5 and between brokers 1 and 2). As mentioned previously, a data log for most recently received data packets must be employed to avoid any looping.

To evaluate a single broker scenario, we execute both subscribers (named A and B) associated with the same broker. First, we evaluate when the broker is running at node two, and in the second case, the broker runs in node seven. It is used the same publication pattern applied to the federated scenario.

The routing process is relatively straightforward by choosing the shortest path between node six and the broker. The results (see Table 3) once again show that the delay is more considerable for the situation when the broker is farther away from the publisher. In addition to that, the aggregate delay for both sub-

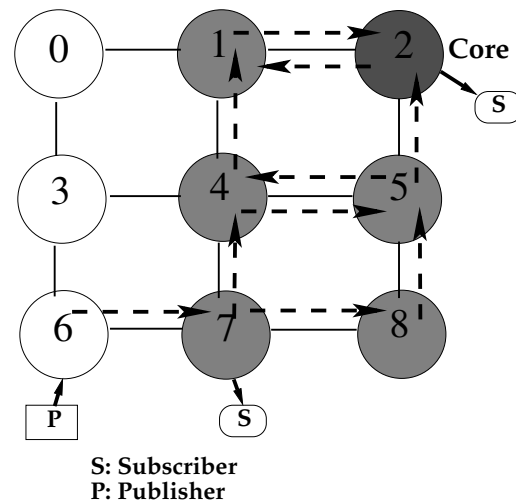


Figure 6: Data packets' routing.

scribers is more extensive when compared to the federated scenario, mainly because the broker has to handle twice the load on average.

7.1 Discussions

The Docker-based Mosquitto instances were shown easy to set up and configure. Given that our solution does not require any changes to the broker, the application based approach was able to act right on the targeted resources. The federation itself is transparent to the brokers, while the virtual infrastructure built around them allows publications to reach their intended subscribers.

Even though the initial proposal assumes an individual mesh for every single topic, we can once again resort to PUMA and lean on its multicast group aggregation approach. Therefore, we should foster solutions that bring topic aggregation to the inner mesh construction and maintenance process. Nevertheless, it would require improvements to the core and mesh announcements so that the largest number of targeted topics can share the same mesh structure.

We might handle federators' scalability issues through proper multi-threading management. One could also pursue a configuration with multiple federator instances associated with a single broker.

Even though we have not employed any SDN or NFV resources, it is left for future work to explore such capabilities. A node entity requires to know its immediate one-hop neighbors in the virtual topology to get things started up. It does not matter where nodes are physically present since they get to communicate to their neighboring nodes. Therefore, there is plenty of space to explore virtual network resources spanning multiple domains/providers.

Table 2: Federated solution: publication delay (publisher at node 6).

Publ.	Subscriber at Node 2	Subscriber at node 7
500	34.64 ± 3.21 ms	18.65 ± 3.2 ms
1000	34.77 ± 3.17 ms	18.61 ± 3.17 ms

Table 3: Centralized solution: publication delay (publisher at node 6).

Publ.	Broker at Node 2		Broker at Node 7	
	Delay (ms)		Delay (ms)	
	Sub. A	Sub. B	Sub. A	Sub. B
500	34.59 ± 3.13	42.47 ± 4.37	18.63 ± 3.2	26.69 ± 4.77
1000	34.58 ± 3.23	42.58 ± 4.48	18.47 ± 3.22	26.42 ± 4.56

8 CONCLUSION

The P/S communication paradigm has paved the way for the development of many IoT applications and platforms. Communication between clients (i.e., publishers and subscribers) can occur asynchronously, allowing many devices connected to the server/broker. Among the available P/S protocols, MQTT plays a vital role in the development of IoT applications.

Nevertheless, in the centralized MQTT architecture, the broker configures itself as a single point of failure, besides being a potential bottleneck. To better deal with such limitations, one could resort to architecture with redundant broker deployment. Anyhow, the emerging environment would require more attention to the configuration and management of the infrastructure.

A protocol for the federation of autonomous brokers is available in the literature. The solution centers on subscribers' meshes, and it lets us build and maintain a routing infrastructure with minimum control overhead. However, the proposal does not include any real implementation. In addition to that, the solution is supposed to require modifications to the inner implementation of brokers.

This work explored an endogenous approach for the federation of MQTT brokers: it proposed the federation of brokers through an application, named federator, playing a supporting role together with standard brokers. The P/S mechanism is itself the primary communication mechanism employed for achieving the federation of brokers.

While standard brokers require no modifications, applications must adhere to the federation protocol. Be that as it may, it could be a little price to pay for the extra degree of freedom when it comes to building a federation infrastructure. With all the currently available virtualized computing and communication resources, one can pick the desired virtual network

topology as needed, all at the application layer. One could conceive the federation process as a new cloud service.

Given that there is no available implementation for the original federation protocol, it is impossible to compare their solution to ours in terms of overall performance. However, the other benefits (e.g., load balancing, increased degree of fault tolerance) outweigh eventual manageable performance issues.

REFERENCES

- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.
- Corp., I. (2020). Distributed publish & subscribe for the internet of things (dps). <https://intel.github.io/dps-for-iot>.
- Docker (2020). Docker container. "https://docs.docker.com/engine/reference/commandline/container/".
- Foundation, E. (2020a). Eclipse mosquito: An open source mqtt broker. "https://mosquitto.org".
- Foundation, E. (2020b). The paho project. "https://www.eclipse.org/paho/".
- HiveMQ (2020). Hivemq's mqtt broker. <https://www.hivemq.com/>.
- Park, J.-H., Kim, H.-S., and Kim, W.-T. (2018). Dm-mqtt: An efficient mqtt based on sdn multicast for massive iot communications. *Sensors*, 18(9).
- Spohn, M. A. (2020). Publish, subscribe and federate! *Journal of Computer Science*, 16(7):863–870.
- Standard, O. (2020). Mqtt version 5.0. <http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.
- SwiftMQ (2020). Swiftmq platform. <https://www.swiftmq.com/>.