Design and implementation of an advanced MQTT broker for distributed pub/sub scenarios

Edoardo Longo[a], Alessandro E. C. Redondi[a]

[a]*Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano, 20134, Italy*

Abstract

MQTT is one of the most popular communication protocols for Internet of Things applications. Based on a publish/subscribe pattern, it relies on a single broker to exchange messages among clients according to topics of interest. However, such a centralized approach doesn't scale well and is prone to single point of failure risks, calling for solutions where multiple brokers cooperate together in a distributed fashion. In this paper, we present a complete solution for a distributed MQTT broker systems. We target several functional primitives which are key in such a scenario: broker discovery and failure recovery, overlay tree network creation and message routing. Moreover, we also focus on the case where multiple topics are present in the system. In such a scenario, a single tree-based overlay network connecting the different brokers may not be the most efficient solution. To cope with this issue, we propose a topic-based routing scheme for MQTT distributed brokers. The proposed solution creates multiple overlay networks in the distributed system, each one linking together only the brokers whose connected clients have interest in the same topics. We implement the complete system as an extension of the popular HiveMQ MQTT broker and perform several experiments to test its performance in scenarios characterized by a different publishers/subscribers configurations as well as number of topics existing in the system.

Keywords: MQTT, distributed pub/sub, topic-based routing

1. Introduction

The Internet of Things (IoT) ecosystem is growing quickly: according to a recent Cisco report [1], IoT connections will be half of the global connected devices and connections by 2023, growing from 33 percent in 2018.

Consequently, communication technologies and protocols used for IoT applications will need to tackle new demands and requirements such as growing traffic volume, extreme low-latency, service complexity, higher quality of user experience, and increasing number of heterogeneous devices. Most of the nowadays IoT applications use the Message Queuing Telemetry Transport (MQTT) protocol at the application layer. Developed by IBM in 1999, MQTT follows a traditional publish/subscribe pattern: clients publish information relative to a particular topic towards a central broker, which is in charge of forwarding it to the interested subscribers. The broker ultimately controls all aspects of the communication between publishers and subscribers, furthermore decoupling them in both space and time. This reason, combined with the great simplicity of the protocol client-side, has contributed to make MQTT the gold standard application-layer protocols for IoT applications. However, relying on a single broker has important drawbacks: indeed, such a centralized design does not scale well considering the massive numbers of IoT devices forecasted in the near future. In addition, it is prone to single point of failure risks on the central broker. For these reasons, several recent works have explored the possibility of creating systems of distributed MQTT brokers, interconnected together and acting as a single entity, with the final goal of ensuring high scalability, elasticity, resiliency to failures and possibly message replication. Such a distributed system of MQTT brokers find use in several application scenarios related to the IoT area, such as:

- 5G Multi-access Edge Computing (MEC): 5G cellular networks have been designed with an inherent support for IoT applications, being able to serve up to 10^6 low-power devices per square kilometre via massive machine type communication (mMTC) technologies as well as using MEC services. The latter will be located in close proximity to the users (e.g., directly at the cellular base stations, as illustrated in Figure 1(a)) providing them with storage, computation and other services (including MQTT broker functionalities), with the final goal of reducing latency as much as possible [2, 3, 4].

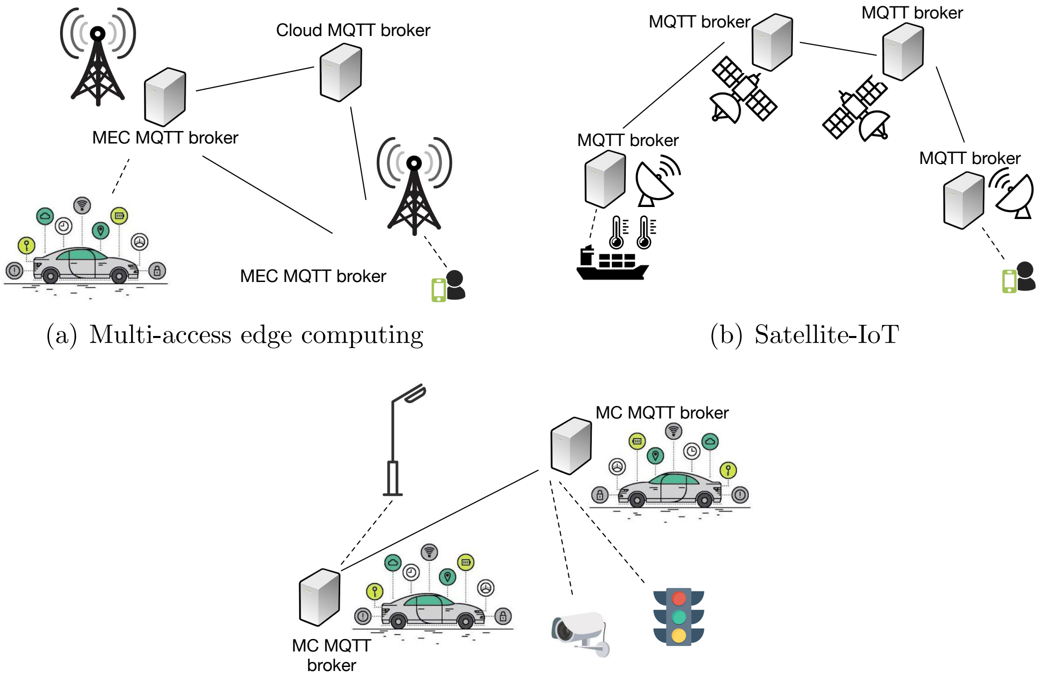
- Satellite IoT: distributed MQTT solutions may play a leading role in future IoT networks based on satellite links, which provide ubiquitous coverage and reliability in places where no other terrestrial technology could [5, 6, 7]. In such a scenario it is common to envision several MQTT brokers communicating among them: as illustrated in Figure 1(b), MQTT brokers may be installed on terrestrial gateways between end devices and the satellites, as well as on the satellites themselves. Recent works already envision an interplay between MEC services and satellite networks based on the MQTT protocol [8]. Managing efficiently such a distributed system of MQTT brokers becomes key, also considering the large delays and unstable links generally involved in satellite communications.

- Micro-clouds (MC): an emerging paradigm in IoT applications consist in developing small data centers which are located very close to the end devices (typically one hop away) and offer communication, sensing, data storage and processing services. Implemented as virtual machines, such miniature clouds may be run on general purpose hardware in close proximity to the IoT devices: e.g., vehicles [9] or *content islands* of things [10] (i.e., group of smart things connected to the same micro cloud) among themselves or with other content islands. Also in such a scenario, the use of the MQTT protocol is becoming predominant [11], motivating the need for distributed broker systems (Figure 1(c)).

Works in these areas that make use of the MQTT protocol generally rely on the concept of *bridging*, an existing feature provided by some existing MQTT broker implementations (e.g. Mosquitto, HiveMQ, CloudMQTT) which allows a broker `B` to connect to another broker `A` as a standard client, subscribing to all or a subset of the topics published by clients to `A`. Unfortunately, such a procedure is extremely time consuming and prone to message loops among brokers: indeed, the existence of a cycle where a message is continuously republished by the participating brokers can quickly deplete a broker's resources, ultimately making it unable to deliver meaningful traffic. Due to the enormous complexity of implementing duplicate detection in distributed scenarios, which would require to keep track of the original producer of every message received and forwarded by any broker, existing solutions require to manually configure the connections between brokers in a static tree-based topology, which lacks adaptivity and robustness.

This work presents and evaluates a complete solution for creating a flexible and efficient distributed system of MQTT brokers. In details, the main features implemented in the proposed solution are:

- *Broker discovery:* in contrast to other available solutions for interconnecting MQTT brokers (e.g., MQTT bridging), which require a static,



(a) Multi-access edge computing

(b) Satellite-IoT

(c) Micro-clouds

Figure 1: Application scenarios where distributed MQTT systems may be applied

a-priori knowledge of all brokers, we propose a mechanism to automatically discover brokers, speeding up the creation and management of the distributed architecture.

- *Overlay creation:* we make efficient use of our previous work named MQTT-ST [12] to create a logical spanning tree among the brokers joining the distributed architecture, thus avoiding the cumbersome task of manually creating a loop-free topology. The spanning tree is created automatically by the brokers, taking into account parameters such as inter-broker latency and computational/memory resources. Furthermore, the tree is robust to broker failures and it is able to reconfigure automatically upon malfunctions.

- *Message routing:* the proposed system allows for either full message replication among the brokers or efficient and targeted delivery thanks to a routing algorithm built on top of the overlay tree which greatly reduces the amount of traffic needed to interconnect clients associated to different brokers.

- *Topic-based optimization:* we also propose mechanisms to cope with the scale and dynamism of networks exchanging messages over a large number of topics. In such a scenario, a single overlay tree determined *a-priori* only by the capacity of the brokers and by the latency of the links among them may not be the optimal choice. Indeed, a broker has often to receive and forward messages about topics to which it is not interested just because it is an intermediate node of the route to reach the interested subscribers. Therefore, in this paper, we take a further step and propose a distributed system of MQTT brokers which supports multiple overlay networks, and consequently routing decisions, one per each topic in the system.

A distinctive feature of our solution is to use in-band signalling as much as possible: indeed, most of the control procedures required to set up and maintain the distributed system are implemented reusing MQTT native mechanisms such as publication messages as well as ping request and response messages. We design and implement the system as a plug-in of the popular HiveMQ MQTT broker[1] and test it in several network scenarios. We compare our solution with a benchmark distributed system using a single overlay network and shows its superior performance. The remainder of this paper is structured as it follows: Section 2 reviews related works, Section 3 introduce the design and implementation of the system, Section 4 describes the overlay tree creation, also considering the topic-based optimization. Section 5 focuses on the brokers' routing protocol, Section 6 provides experimental results and Section 7 concludes the paper.

2. Related Works

Efficient message dissemination in distributed pub/sub systems has been the subject of several research activities in the last 20 years [13, 14, 15, 16, 17, 18]. One common feature in such systems is the existence of a so called overlay network, which allows nodes to reorganize in a virtual network on top of the existing physical network infrastructure.

[1] https://www.hivemq.com/

In these fully decentralized implementations of the topic-based pub/sub systems, nodes (corresponding to brokers) forward their messages in a peer-to-peer overlay network fashion.

Similarly to our system implementation, in PADRES [13], a special advertisement is added to the common publications and subscriptions messages. These advertisements avoid message flooding in the network, allowing the creation of routing trees along which subscriptions are propagated. PADRES also store routing data in Overlay Routing Tables (ORT) that are used to disseminate advertisements. In addition, subscriptions and publications routing work according to respectively Subscription Routing Tables (SRT) and Publication Routing Tables (PRT). Improvements in the routing paths are carried by *Minimum Topic-Connected Overlay (Min-TCO)* [19] networks and further developments [20, 21], where all the nodes with shared topics are connected minimizing the possible number of edges. Therefore, a message published on the topic t reaches all the nodes interested in t being forwarded only by nodes interested in t. However, in these cases, the minimisation problem takes into account only the number of hops and the delay on the link is not considered.

Teranishi et al.[22] introduce the concept of location awareness in topic based pub sub for IoT applications. In this case, the messages are directly forwarded if publishers and subscribers are in the same physical network, instead of being flooded to every peer of other networks.

Thanks to the enormous success of such a protocol in IoT applications, recently some attention has been given to the problem of building and operating a distributed system of MQTT brokers. Some works focused primarily on vertical clustering, where the single broker is replaced by many virtualised broker instances running behind a single endpoint, typically a load balancer [23][24]. These approaches introduce the concept of multiple brokers cooperating with each other, although the broker cluster is seen as a single centralised entity from the perspective of clients.

Pure MQTT broker distribution is introduced in Banno et al. in [25]: the work in [25] proposes ILDM (Internetworking Layer for distributed MQTT brokers), in which heterogeneous brokers are connected to each other through specific nodes, placed between client and broker, and messages are exchanged through flooding. However, the manual configuration of MQTT bridges has two main disadvantages: firstly, similar to the wiring of switches in small and medium-sized companies, it can become a very confusing task with a high possibility of accidentally creating duplicate connections, especially in large topology. Second, by imposing a loop-free static topology between brokers, adaptivity and robustness to failures are completely lost.

A different approach is given in [26], where authors propose the use of a Software Defined Networking (SDN) controller to collect information on clients and their relative pub/sub topics, in order to create per-topic multicast groups to minimise data transfer delay. In [27] and [28] authors propose to interconnect MQTT brokers dynamically, making it possible to change the configuration of the topology at run-time through specific MQTT messages transmitted by a trusted centralised entity. Similarly, the work in [29] creates a distributed broker network by utilising an external agent to control the status of each broker. Upon any change in the broker network configuration (broker failure, increase in latency, etc.) the clients reconnect to a new broker, based on information retrieved by the monitoring agent. Such an approach enables client mobility, dynamic broker provisioning, and broker load balancing.

A new technique that allows the MQTT protocol to cope with distributed brokers for the new IoT challenges was to apply the spanning tree protocol over a network of brokers in order to create a loop-free topology between more brokers [12]. A second example of such application is D-MQTT [30]: this work proposes a distributed version of MQTT, based on the mosquitto broker, where multiple brokers connect with each other through a spanning tree topology, allowing clients to communicate even if originally associated to different brokers.

To the best of our knowledge, this work is the first distributed MQTT system that uses topic-based overlay networks for efficient message dissemination.

3. System Overview

The main goal of this work is to design an effective solution to support distributed MQTT brokers, practically applying it in a realistic scenario. Although the solution reported in this paper could be applied to any MQTT broker implementation, here we take the popular HiveMQ MQTT broker as a starting point and we modify it to support a distributed scenario. The motivation of this choice lies in the very simple management of HiveMQ, which offers a free and open source Java SDK to modify the broker behaviour via plugins. The plugin SDK provides to the developer several callbacks, which can be linked to user-defined logic easily. Conversely, other popular broker implementations such as Mosquitto, require to directly work on the broker source code and are consequently much more complex to modify.

We therefore develop a plugin for turning a single centralized broker into an entity that could easily work in a distributed scenario. The plugin works according to the finite state machine illustrated in Figure 2:

1. *Discovery phase*: Upon startup, the broker is in the `DISCOVERY` state, in order to become aware of other brokers ready to join the distributed system. In details, first each broker joins an IP multicast group. Then, it periodically transmits to the multicast group a specific UDP discovery message, containing its own IP address as well as the port over which the MQTT broker process is listening. At the same time, each broker listens for incoming discovery message from other brokers for a pre-defined period of time T_d. At the end of T_d, a broker knows the addresses and ports of all active brokers in the group. It stops transmitting discovery messages, keeping the receiver process active for other brokers willing to join the distributed system in a subsequent phase. The broker moves then to the `OTC` state. Note that, regardless of the current state, the reception of a discovery message from a new broker causes the system to restart from the `DISCOVERY` state. This allows a broker to join the distributed system at any time. This is the only step of our system which requires an out-of-band (non MQTT specific) message exchange.
2. *Overlay tree creation:* When the discovery phase ends, each broker has knowledge of all other brokers in the distribution system. Then, the plugin enters in the `OTC` state, which takes care of executing a version of the Spanning Tree Protocol (STP) adapted for the particular scenario, so that all brokers agree on a loop-free topology that serves as an overlay network. More details on such a process are given in Section 4.
3. *Message routing:* When the overlay network is created, the plugin moves to the `RUNTIME` state, that takes care of processing events (publications and subscriptions) arriving at the broker and routing them to other brokers connected. Section 5 details this process.
4. *Failure recovery:* Upon a broker failure, the plugin handles the situation by to re-establish the routing tree. In details, the broker detecting a socket error transmit to all other brokers in the tree a specific MQTT message used to restart the tree construction from scratch. The message is published over the special MQTT topic `$TC` (Topology Change),

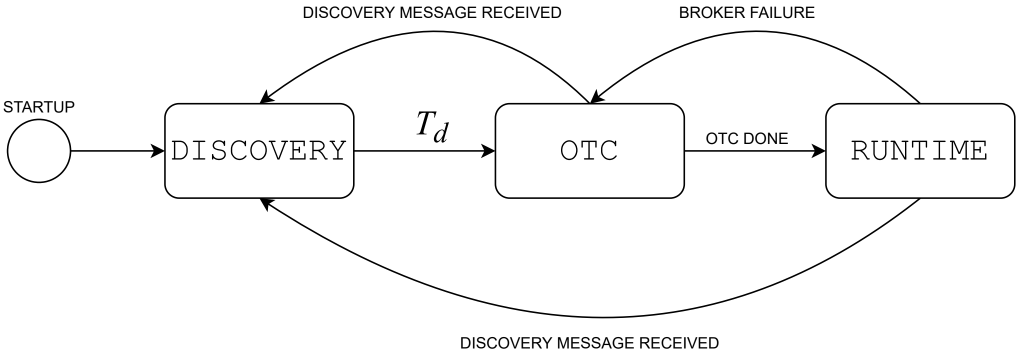


Figure 2: System State Machine

and has the same goal of a topology change message in the STP protocol. Any broker receiving such a message restart the root selection procedure, which eventually will converge to a new tree.

As we shall explain in the following, the overlay tree creation and the routing process may also be specialized on a per-topic basis.

4. Overlay tree creation

After the discovery step ends, all brokers in the distributed system know each other's broker IP address and MQTT listening port. This allows to establish an underlying fully connected topology of brokers, which serves as the basis for the distribution system. In details, each connection between two brokers is implemented as an MQTT bridge, which can be specified in a configuration file of the broker process. Each bridge connection requires to specify the IP address and port of the remote broker, which result from the discovery phase.

To avoid harmful message loops in the distributed system, the creation of an overlay tree network over the fully connected topology specified in the configuration file is needed. For the task at hand, we adapt our previous work [12] (originally developed for Mosquitto brokers) to HiveMQ. The process, known as MQTT-ST, takes inspiration from the well known Spanning Tree Protocol used for layer 2 switches, according to a three-steps process detailed in the following. To further specialize the creation of the overlay to the topics that are used in the system, we also propose a topic-based overlay creation process. In such a way the system supports multiple overlays, and consequently routing paths, according to the number of topics that are active in the system. This helps in avoiding routing messages belonging to specific topics through brokers which are not interested in such topics, with a twofold benefit: minimizing both the end to end latency as well as the number of messages forwarded among brokers. For the sake of clarity we use the term *Single overlay* to refer to the mechanisms put in place to setup a tree topology connecting all brokers, regardless of the topics used in the system. Conversely, we use the term *Topic-based overlay* to refer to the case where multiple overlays are present in the system, one connecting only the brokers interested to a specific topic.

4.1. Signalling phase

In order to maintain a connection towards a broker, MQTT requires each client to periodically transmit specific messages, known as PINGREQ messages. The periodicity of such transmission is regulated by the Keep Alive parameter, specified upon the first connection. In case a broker does not receive any message from a client within the keep alive interval (or the client does not receive the corresponding PINGRESP) answer, the connection is closed from one or the other side.

4.1.1. Single overlay

The proposed system reuses the concept of the PINGREQ message, which plays the role of Bridge Protocol Data Unit (BPDU) messages used in the STP protocol. In that case, BPDUs are broadcasted periodically by every switch and contain information used to compute the root node of the spanning tree and the best path to the root. In our case, PINGREQ messages are enriched with additional information, including the IP address of the current selected root broker, a root path cost P (in terms of latency) and the broker capability value C. The latter two fields are used for path computation and root selection, respectively and will be detailed later. Upon reception of a PINGREQ message, a broker replies with a PINGRESP message. Such exchange allows the originating broker to estimate the RTT to the destination.

4.1.2. Topic-based overlay

In this case, we take inspiration from PVST+ (Per VLAN Spanning Tree Plus), a protocol developed from Cisco to enable multiple VLANs on the same network of layer 2 switches. This is obtained by broadcasting multiple BPDUs, one per VLAN, where each BPDU is tagged with the information on the corresponding VLAN. In our case, the main goal is for one broker to broadcast to others its topics of interest, corresponding to either local publications or subscriptions. Therefore, upon receiving the first subscription or publication from a client on a new topic t, the broker periodically broadcasts a topic-specific PINGREQ message to notify other brokers about its interest in the topic. The message contains the topic name t, the broker capability value C as well as the path cost P_t for reaching the current root of the overlay for topic t. Furthermore, the topic-specific PINGREQ message contains two additional fields: (i) the number of topic-specific overlay trees for which the broker is currently root node, R and (ii) the total number of clients connected C. Such additional metrics will be used to compute in the root selection phase to better balance the load among brokers. Differently from the single overlay case, upon reception of a topic-specific PINGREQ, a broker replies with a PINGRESP only if it has at least one client interested in the same topic (either as publisher or subscriber). In this case, the PINGRESP message contains just the name of the topic t and it is used by the requesting broker to keep track of other brokers interested in that topic. After T_t seconds from the transmission of the topic-specific PINGREQ, a broker stores in memory the IP addresses of the replying brokers. This information is used to perform a specialized root selection and path computation process only on the subset of brokers interested in the advertised topic.

4.2. Root selection

The root broker plays a crucial role in the broker tree, as it is the relay node for all traffic and it is, therefore, subject to an increased computational load. Indeed, selecting a broker with poor or overloaded resources may result in poor overall performance.

4.2.1. Single Overlay

In STP, the root is selected based only on its identifier, which does not suit well the scenario under consideration. In this case, each broker transmits to all other brokers a PINGREQ message containing (i) the IP address of the currently selected root node, (ii) the current path cost to the root P and (iii) the current broker available resources, in terms of memory M and CPU L. Upon startup, each broker sets itself as the root: therefore the path cost P is equal to 0. Once each broker receives the PINGREQ from all other brokers, the root of the tree can be elected in a unique way: in particular, the root broker is selected according to the capability value C, defined as:

$$C = \alpha L + \beta M \tag{1}$$

where α and β are tuneable conversion parameters. In case of a tie, the broker with the lowest IP address is selected as root.

4.2.2. Topic-Based Overlay

In this case, one overlay per topic is created. Therefore, a broker may be participating in multiple overlays, with different roles (e.g., root/intermediate node/leaf). Taking this into account, we developed two different techniques, each one following a different policy to select the root of each topic tree:

- *Capacity Based*: The first algorithm follows the same root selection policy of the single overlay case: the root is selected based on the capacity of the brokers involved in the formation of the tree; for every tree, the broker with the highest capacity is always selected as root. Similarly as before, the capacity is calculated as in Equation 1.

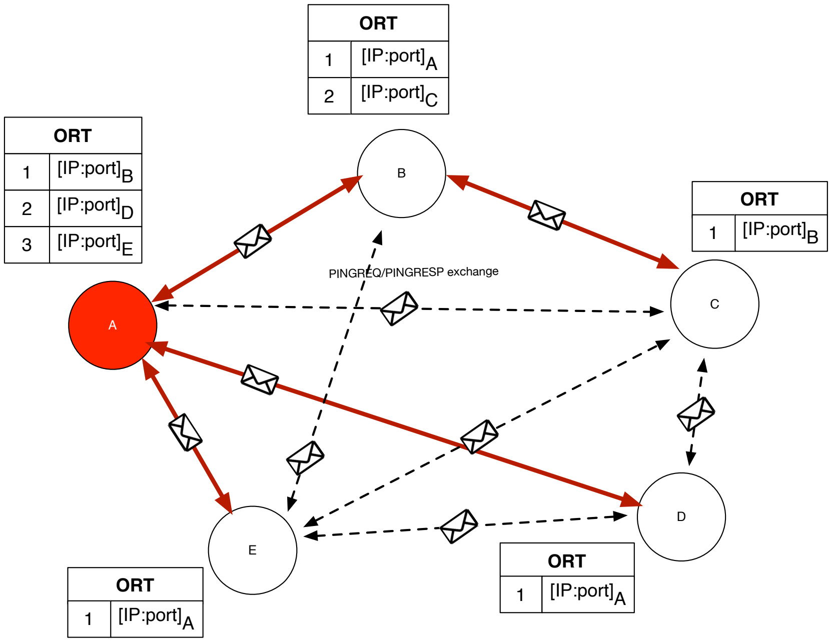
- *Root Balance based*: The purpose is to *balance* the roots of the trees among the available brokers. Considering only brokers' capacities, the most powerful node may be the root of multiple spanning tree, leading to bad load balancing. In order to balance the selection of roots among the available brokers, for each tree we selected as a root the broker having the lowest *Root Balancing (RB)* value that is computed in the following way:

  $$RB = \gamma R + \delta C$$

  where R is the number of trees in which each broker is the root, and C is the number of clients connected to each broker; γ and δ define the *weight* assigned to the two parameters.

4.3. Path computation

Regardless of the scenario under consideration (single overlay or topic-based overlay), brokers keep on transmitting periodically PINGREQ messages. Each broker receiving a PINGREQ replies with a PINGRESP message: this allows to estimate the round trip time (RTT) between two brokers.



ORT	
1	[IP:port]$_B$
2	[IP:port]$_D$
3	[IP:port]$_E$

ORT	
1	[IP:port]$_A$
2	[IP:port]$_C$

ORT	
1	[IP:port]$_B$

ORT	
1	[IP:port]$_A$

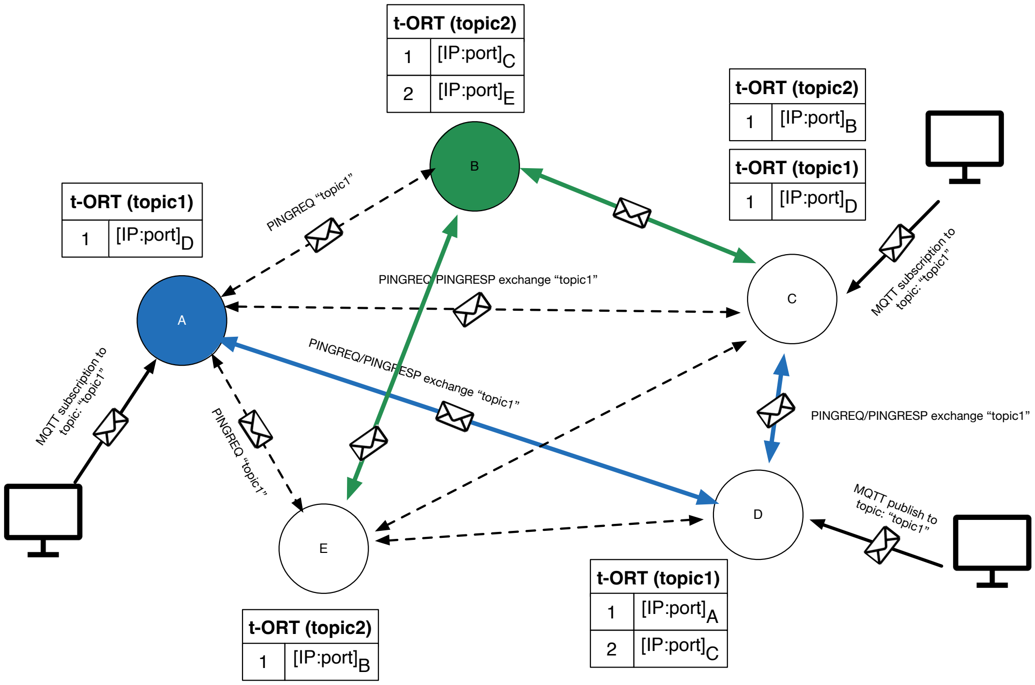
ORT	
1	[IP:port]$_A$

Figure 3: Overlay Routing Table (ORT) creation

At each new PINGREQ message, the current path cost P is set equal to the RTT to the currently selected root. Each broker can therefore select what is the best neighboring node for every overlay it is involved in, in terms of total latency, to reach the specific root and therefore creating the overlays. Concurrently, each broker also maintains a global Overlay Routing Table (ORT) and one topic-specific ORT (t-ORT) for each topic it is interested into. Such tables contain the IP addresses and ports of the broker process on neighboring brokers, as illustrated in Figure 3 and 4. The PINGREQ/PINGRESP exchange is also used to identify potential broker disconnection from the system. In particular, if the PINGRESP message from a bridged broker is not received within T_o seconds from the transmission of hte PINGREQ, a broker marks the target broker as disconnected and transmits a discovery message to restart the tree computation process.

5. Routing Protocol

Other proposals for MQTT distributed systems [12, 25, 31, 32] rely on the use of flooding for disseminating messages among brokers. Flooding is


PINGREQ "topic1"
B
A
PINGREQ/PINGRESP exchange "topic1"
C
MQTT subscription to topic: "topic1"
PINGREQ/PINGRESP exchange "topic1"
MQTT subscription to topic: "topic1"
PINGREQ "topic1"
PINGREQ/PINGRESP exchange "topic1"
E
D
MQTT publish to topic: "topic1"

t-ORT (topic2)	
1	$[\text{IP:port}]_C$
2	$[\text{IP:port}]_E$

t-ORT (topic2)	
1	$[\text{IP:port}]_B$

t-ORT (topic1)	
1	$[\text{IP:port}]_D$

t-ORT (topic1)	
1	$[\text{IP:port}]_D$

t-ORT (topic1)	
1	$[\text{IP:port}]_A$
2	$[\text{IP:port}]_C$

t-ORT (topic2)	
1	$[\text{IP:port}]_B$

Figure 4: Topic-based Overlay Routing Table (t-ORT) creation

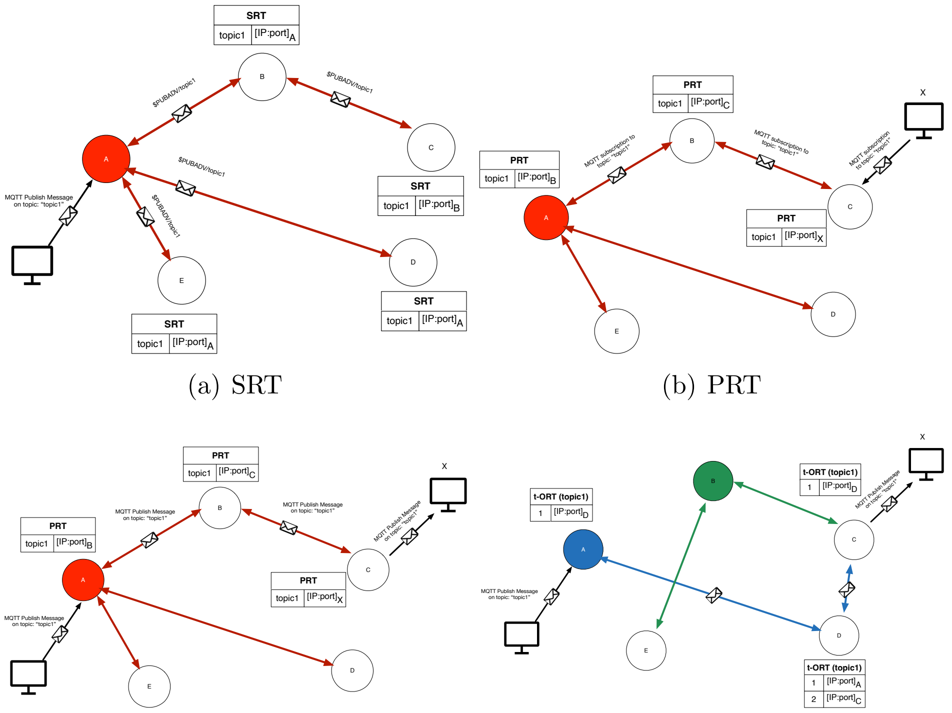
indeed an important mechanism also in our proposal as it find use in both the single-overlay and topic-based overlay scenarios.

5.1. Single overlay

The single overlay tree specified in the ORT tables connects all broker in the distribution system. Upon a publication on a specific broker, flooding may be used to deliver such message to all other brokers without creating harmful message loops. This feature is convenient in some cases, e.g., full message replication among the brokers. However, when full replication is not required, the single overlay tree has the drawback of forwarding messages also to brokers which may not be interested in the publication topic. To improve on such aspect, we implement in our system a routing protocol to route messages only towards interested brokers. The protocol is an adaptation of the one proposed in PADRES [13]: There, subscriptions are routed according to a Subscription Routing Table (SRT), which is created according to the advertisements sent by the brokers. Such an advertisement, typical of content-based pub/sub protocols, is flooded to all brokers to populate routing tables and ensure that there is a path from any publisher to any subscriber with a potentially matching subscription. We adapt such a mechanism to our system: in particular, when a broker receives a publication message on a new topic for the first time in its life cycle, it floods it to all its neighbouring brokers (i.e., the ones in its ORT) on the topic `$PUBADV/<topic>` where `topic` is the original subscription topic. Each broker receiving such an advertisement updates its Subscription Routing Table (SRT), storing the next hop a subscription should take in order to be forwarded to the broker having publishing clients matching such a topic. As an example, Figure 5(a)) shows (i) the SRTs of all brokers after the flooding of the advertisement message from broker A. Every broker also has a Publication Routing Table (PRT), which keeps track of clients subscribed and the corresponding topics. PRTs are used to forward publications matching topics to subscribers, and are a fundamental component of topic-based pub/sub system. Here, PRT can also be used to track subscriptions coming from different brokers. As an example, Figure 5(b) shows a client connected to broker C with IP address X which subscribes on `topic1`. Broker C has therefore an updated entry in its PRT. Since the subscription topic is also existing in the SRT, the subscription is routed following the SRT to brokers B and A, which can update their own PRTs accordingly. Finally, Figure 5(c) shows a new publish message on `topic1` from a client connected to broker A can be routed to the client subscribed on broker C in an efficient way following the PRTs, avoiding unnecessary message flooding to all brokers in the tree. Note that, in case a subscription is received by a broker before the corresponding advertisement is created, the broker stores the subscription locally in the PRT as per the legacy MQTT logic. At each new advertisement message received and SRT update, each broker checks again its PRT and forwards any matching subscription following the SRT.

5.2. Topic-based overlay

The single overly routing, although providing improvements compared to flooding over the ORT, still suffers from an important drawback. As an example in Figure 5(c) the publication from broker A needs to pass from broker B in order to reach broker C, although B is not interested in such a topic. The topic-based overlay proposed here solves this problem by design, since each topic corresponds to an overlay tree formed by only those brokers interested in it. Therefore, upon a publication, a broker uses the t-ORT



(a) SRT

(b) PRT

(c) Publish with SRT/PRT

(d) Publish with t-ORT

Figure 5: (a) SRT population after advertisement flooding, (b) PRT update upon client subscription, (c) Message routing on the single overlay via SRT and PRT tables, (d) message routing using flooding over different topic-based overlays

corresponding to the topic to flood the publication message to all brokers belonging to the t-ORT. This is illustrated in Figure 5(d).

6. Experimental results

We implemented the algorithms explained in Section 4 and 5 in the form of plug-ins of the widely-known HiveMQ broker. We use the HiveMQ CE version[2], a Java-based open-source broker that supports up to MQTT version 5. It also provides an extension framework[3] allowing developers to create custom extensions to add and/or modify the HiveMQ broker functionalities

[2] https://www.hivemq.com/developers/community/

[3] https://www.hivemq.com/docs/hivemq/4.5/extensions-javadoc/index.html.

Table 1: Most relevant experiment parameters

Parameter	Value
Number of brokers	5
Propagation delay broker-broker	random
Bandwidth broker-broker	10Mbps
Broker RAM upper limit	2 GB
Number of CPU per broker	1
Number of published messages	500
Number of publishers	5
Number of subscribers	20
Message publishing rate	1 msg/s
Message publishing rate (stress)	10 msg/s
Number of topics	3, 30, 60
Topics name	*topic-tree/*<`random`>/
Message payload size	10^3, 10^4, 10^5 B

for their specific infrastructure.

For the test scenario, we deployed five brokers in the system in a full-meshed physical topology. The network is created using the BORDER framework [33], an extension of Mininet[4] specifically built to interconnect MQTT brokers. Each broker runs inside a Docker container built on top of the HiveMQ CE image[5]. Subsequently, the brokers are connected through a Timing Compensated High-Speed Optical Link (TCLink), with adjustable link delay and link bandwidth parameters. Thanks to the BORDER framework, the brokers run isolated as in a sandbox, without any interference on the access of the resources. For each broker, we allocated a core of the host machine [6] and 2GB RAM.

The following details the parameters used for the tests, also summarised in Table 1:

- *Brokers*: we create 5 brokers in the system, in a full-meshed physical topology. The capacity of each link is fixed to 10 Mbps and the propagation delays are chosen randomly from a normal distribution with

[4] http://mininet.org

[5] https://hub.docker.com/r/hivemq/hivemq-ce

[6] Intel(R) Xeon(R) CPU E5-1660 with 16 CPU @ 3.00GHz

$\mu = 10$ms and $\sigma = 2.5$ ms.

- *Publishers*: we connect 20 MQTT publishers to the system, each one publishing to one topic chosen randomly among a variable number topics. Each publisher publishes a total of 500 messages, 10 message per second, each containing a timestamp and a random string of 1024 bytes.
- *Subscribers*: we connect 5 MQTT subscribers to the system, each one subscribing to one of the publishing topics.
- *Topics*: According to the experiment, we create respectively 3, 30 or 60 topics of interest. Each subscriber in the system chooses randomly among a set of available topics; similarly, each publisher publishes randomly on one of them.

To evaluate different traffic scenarios, we consider three different *locality* cases:

- *Locality 100%:* this is the maximum degree of locality, meaning that all publishers and all subscribers are connected to the same broker.
- *Locality 0%:* conversely, this case represents the scenario with the minimum degree of locality, where all publishers are connected to one broker and all subscribers are connected to a single, different broker.
- *Locality 50%:* we also test an intermediate case, where publishers and subscribers are randomly distributed among the 5 brokers. In this case, we repeat the test 10 times changing each time the distribution of publishers and subscribers and we compute the average result.

For every experiment we compare the proposed single overlay and topic-based overlay extensions to a *benchmark* solution obtained by flooding all publications over the single overlay.

6.1. Average end-to-end delay

First, we analyse the average end-to-end delay, i.e., the average amount of time elapsed between the publication of a message and its reception by a subscriber. As one can see in Figure 6, we can observe that the proposed topic-based overlay extensions lead to a consistent decrease in the end-to-end delay compared to both the benchmark case as well as the single overlay tree,

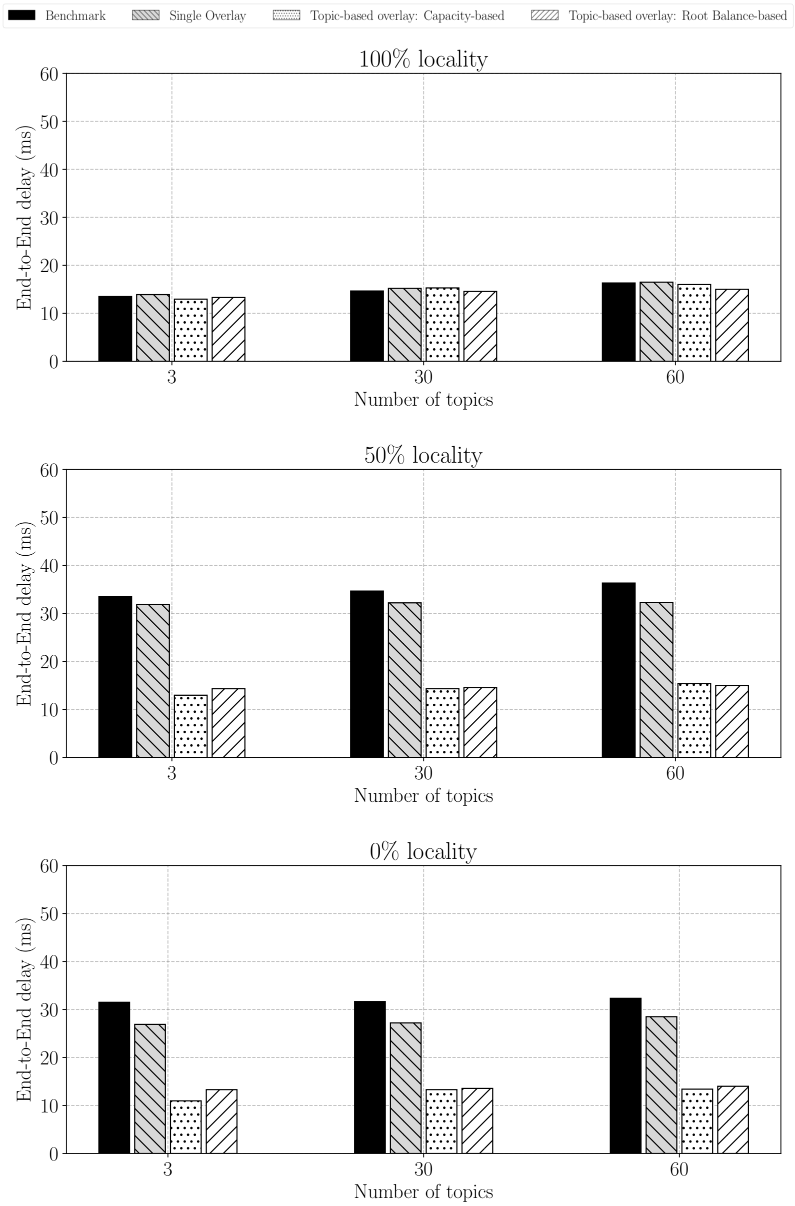


Figure 6: Average message End-to-End delay varying the number of topics, for 100, 50 and 0% locality

especially in cases where publishers and subscribers are distributed among the cluster, e.g. 50% locality. In this case, the topic-based overlay extension tree allows to reduce the message delay by more than a half compared to the other two cases; this is also true for the case of 0% locality, when using the topic-based optimization allows to connect the two brokers hosting publishers and subscribers directly. As regards the two topic-based overlay extensions (capacity and root-balance), they perform similarly, with the capacity-based version showing a slightly better delay response. We can also see how in those cases where the benefits of specific overlay routing tree are less evident (e.g. 100% locality), the end-to-end delay of the of topic-tree based versions stays in line with the benchmark, without performance degradation.

6.2. Traffic overhead

In a distributed scenario is important to take into account the additional traffic exchanged by the brokers. This traffic is composed by two main parts: (i) signalling overhead, which is the traffic created by the protocol mechanisms, and (ii) intra-broker publishing communication, which corresponds to the publications forwarding inside the cluster of brokers.

We start by analyzing the signalling overhead, illustrated in Figure 7. For the benchmark case, the traffic comprises standard MQTT control messages among brokers (mainly connect messages and MQTT ping requests/responses). For the single overlay case, the traffic overhead includes also discovery messages, ping requests/responses used as BPDU, publication advertisements forwarding and subscriptions forwarding. Finally, for the *topic-based overlay* network, the overhead traffic includes all aforementioned components and the per-topic overlay network control messages (again, implemented as PINGREQ/PINGRESP exchange). Clearly, since in this latter case a broker transmit one PINGREQ per every topic it is interested into, the overhead traffic increases as the number of topics in the system grows.

As one can see, in the benchmark and single overlay cases the signalling traffic is negligible compared to the intra-broker traffic. Moreover, it does not depend on the number of topics existing in the system. Conversely, for the topic-based overlay scenario, the overhead traffic is mainly influenced by the messages needed to create the t-ORT tables and thus very susceptible by the number of topic, increasing linearly with them. With only 3 topics, the *topic-based overlay* version has only a 30% increase compared to the benchmark; with 60 topics the increase is of the 1500%.

At the same time, the intra-broker traffic (green background bars in Figure 7) greatly benefits from the use of topic-based overlay networks, because messages are routed only to the interested brokers. In particular, it becomes null for the 100% locality case. Indeed, in this case no messages are exchanged among brokers, since subscribers and publishers are connected to the very same broker.

Note that, considering the total traffic exchanged (intra-broker publications and signalling traffic), both the single overlay and the topic-based overlay extensions always greatly outperform the benchmark case. It can be seen that the topic-based overlay is not always the best options: for example, with a low number of messages (500) and low payload (1000 bytes), the topic-based overlay version is penalized compared to the single overlay one. The performance reduction is only due to the signalling traffic overhead: the impact of such overhead can be amortized over an higher number of messages exchanged and considering longer system lifetimes.

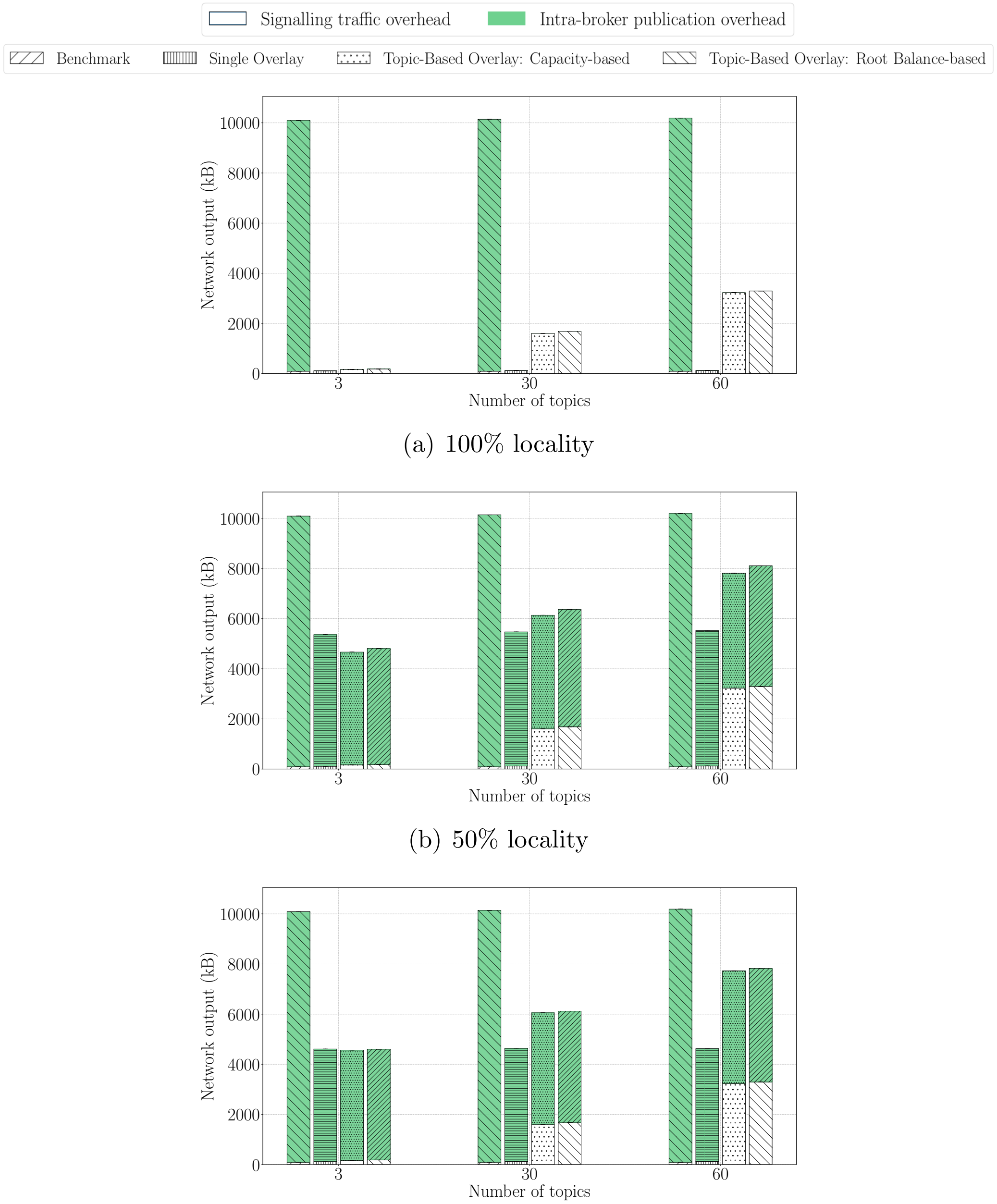
In Figure 8, we keep the number of messages published and the active topics constant (60 topics and 500 messages) but we increase the size of the publication messages from 10^3 to 10^5 bytes. As we can see, here the signalling overhead impact is almost negligible, having the intra-broker traffic the biggest weight in the system. In this case, the topic-based overlay protocol shows the expected improvements compared to the version that uses only a single overlay network.

6.3. Resource usage

We also evaluate the amount of resources (i.e., CPU and RAM) consumed by the brokers running the extensions[7]. From the results, it can be observed that all the extensions have a similar behaviour under the CPU and RAM point of view since no relevant differences have been found with the configuration used above. In order to have an all-around vision for resources consumption, we tested the extensions in stressful conditions. For this reason, we increase the publish rate from 1 to 10 messages per second.

As one can see in Figure 9, in the stress test, as expected, the topic-based overlay algorithms had higher consumption of CPU with respect to the benchmark. Particularly, as the number of the topics grows, and the number of trees does as well, brokers need more CPU to handle all the message routing to the topic trees. The CPU results in an average increase of 22%; this consistent growth, however, is shown only in an upper limit

[7] For this case we consider only the 50% locality configuration, however, no significant difference were noted with the other configurations.



(a) 100% locality

(b) 50% locality

(c) 0% locality

Figure 7: Traffic overhead: average bandwidth used by the overlay networks based brokers against the benchmark for 100, 50 and 0% locality

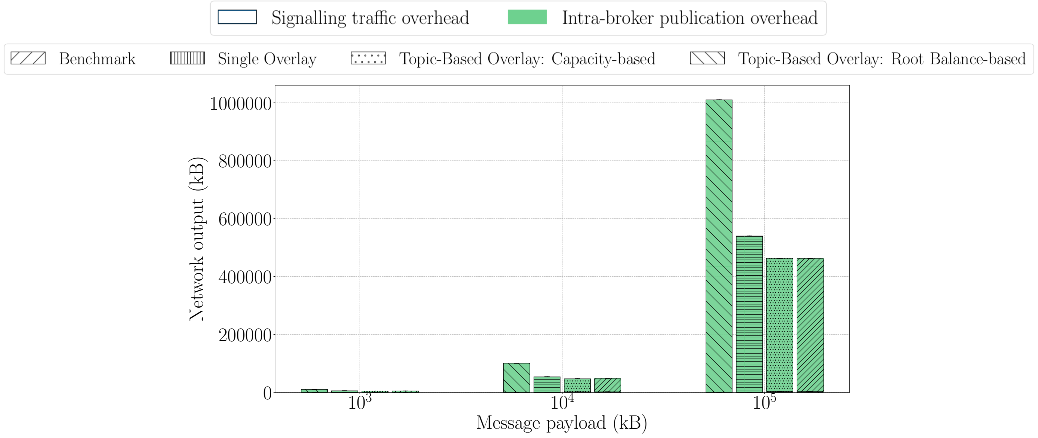


Figure 8: Traffic overhead: impact of message size in the traffic overhead for the proposed protocols.

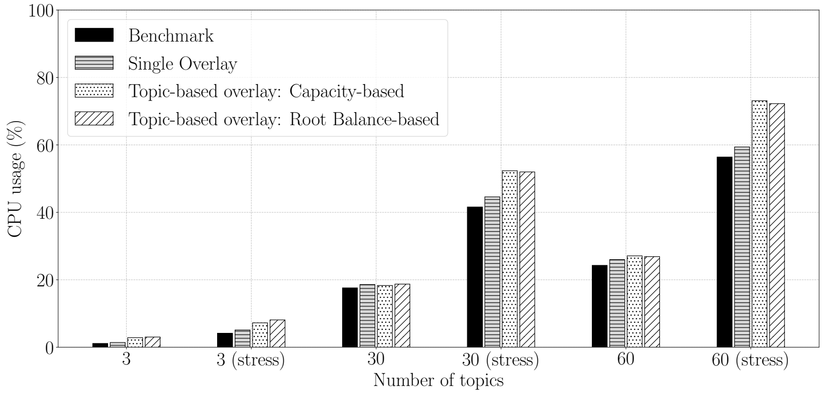


Figure 9: Average CPU usage by the ensemble of brokers in the test scenario and in stress conditions.

case for an IoT application (usually low-rate based). Note also that only one CPU core was allocated for each broker, which results to be highly stressed. Regarding RAM consumption, we do not find significant increases in the stress test, showing that the additional data structures used for storing the ORT, PRT, SRT and t-ORT tables perform efficiently.

6.4. Convergence and repair time

We also report the results obtained for what concerns two specific time measurements:

Table 2: Convergence and repair times

	Avg. value	Std. deviation
Convergence Time [sec.]	19.5	0.02
Repair Time [sec.]	59.1	0.01

- *Convergence time:* from the system start-up instant to the time where the last broker compute the spanning tree.

- *Repair time:* the time elapsing from the hard disconnection of one randomly selected broker from the system to the the instant when a new spanning tree is computed.

We run each experiment 5 times and we average results, reported in Table 2. As one can see, we obtained good and robust setup and recovery performance, as the standard deviation of the time measurements obtained is very low. Clearly, the average values obtained are tightly coupled to the timer T_d used in the discovery process as well as the timeout T_o used for identifying a broker disconnection, which can be optimized in future work for even shorter convergence and repair times.

7. Conclusion

Novel application scenarios for IoT applications will require well-known and centralized protocols such as MQTT to move towards distributed architecture able to support efficient and timely message delivery. This paper presents a step in such a direction, with the proposal of several extensions to turn a legacy MQTT broker into an entity able to cooperate with other brokers in a distribution system. We tackle the problem of automatic broker discovery and failure recovery, overlay network construction in order to avoid message loops among brokers and propose different routing services to optimize the message delivery. Two main approaches are considered: the construction of a single overlay network coupled with a per-topic routing optimization or, conversely, the construction of multiple topic-based overlay networks where messages can be flooded efficiently among brokers sharing the same interests. We explain in details the construction of such extensions and implement them on the popular HiveMQ MQTT broker. We show the performance obtained by our proposal in several scenarios, highlighting a trade-off between the two approaches in terms of average end-to-end delay obtained and overhead control traffic produces to maintain the overlay networks. Future research directions will target further system optimization, such as, e.g., the aggregation of different topic-based overlay trees in a single one if the participating brokers are the same.

References

[1] U. Cisco, Cisco annual internet report (2018–2023) white paper, Cisco: San Jose, CA, USA (2020).

[2] O. Salman, I. Elhajj, A. Kayssi, A. Chehab, Edge computing enabling the internet of things, in: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), IEEE, 2015, pp. 603–608.

[3] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, T. Taleb, Survey on multi-access edge computing for internet of things realization, IEEE Communications Surveys & Tutorials 20 (4) (2018) 2961–2991.

[4] A. E. Redondi, A. Arcia-Moret, P. Manzoni, Towards a scaled iot pub/sub architecture for 5g networks: the case of multiaccess edge computing, in: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), IEEE, 2019, pp. 436–441.

[5] R. Soua, M. R. Palattella, T. Engel, Iot application protocols optimisation for future integrated m2m-satellite networks, in: 2018 Global Information Infrastructure and Networking Symposium (GIIS), IEEE, 2018, pp. 1–5.

[6] M. R. Palattella, R. Soua, A. Stemper, T. Engel, Aggregation of mqtt topics over integrated satellite-terrestrial networks, ACM SIGMETRICS Performance Evaluation Review 46 (3) (2019) 96–97.

[7] R. Soua, M. R. Palattella, A. Stemper, T. Engel, Mqtt-mfa: a message filter aggregator to support massive iot traffic over satellite, IEEE Internet of Things Journal (2021).

[8] M. Luglio, M. Marchese, F. Patrone, C. Roseti, F. Zampognaro, Performance evaluation of a satellite communication-based mec architecture for iot applications, IEEE Transactions on Aerospace and Electronic Systems (2022).

[9] F. Hagenauer, T. Higuchi, O. Altintas, F. Dressler, Efficient data handling in vehicular micro clouds, Ad Hoc Networks 91 (2019) 101871.

[10] P. Manzoni, E. Hernández-Orallo, C. T. Calafate, J.-C. Cano, A proposal for a publish/subscribe, disruption tolerant content island for fog computing, in: Proceedings of the 3rd Workshop on Experiences with the Design and Implementation of Smart Objects, 2017, pp. 47–52.

[11] R. Venanzi, B. Kantarci, L. Foschini, P. Bellavista, Mqtt-driven sustainable node discovery for internet of things-fog environments, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–6.

[12] E. Longo, A. E. Redondi, M. Cesana, A. Arcia-Moret, P. Manzoni, Mqtt-st: a spanning tree protocol for distributed mqtt brokers, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.

[13] E. Fidler, H.-A. Jacobsen, G. Li, S. Mankovski, The padres distributed publish/subscribe system., in: FIW, Citeseer, 2005, pp. 12–30.

[14] R. Baldoni, R. Beraldi, L. Querzoni, A. Virgillito, Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA 50 (4) 444–459. `doi:10.1093/comjnl/bxm002`.

[15] A. Majumder, N. Shrivastava, R. Rastogi, A. Srinivasan, Scalable content-based routing in pub/sub systems, in: IEEE INFOCOM 2009, IEEE, 2009, pp. 567–575.

[16] J. L. Martins, S. Duarte, Routing algorithms for content-based publish/subscribe systems, IEEE Communications Surveys & Tutorials 12 (1) (2010) 39–58.

[17] G. Siegemund, V. Turau, K. Maâmra, A self-stabilizing publish/subscribe middleware for wireless sensor networks, in: 2015 International Conference and Workshops on Networked Systems (NetSys), IEEE, 2015, pp. 1–8.

[18] V. Turau, G. Siegemund, Scalable routing for topic-based publish/subscribe systems under fluctuations, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2017, pp. 1608–1617.

[19] G. Chockler, R. Melamed, Y. Tock, R. Vitenberg, Constructing scalable overlays for pub-sub with many topics, in: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC '07, Association for Computing Machinery, New York, NY, USA, 2007, p. 109118. doi:10.1145/1281100.1281118.
URL https://doi.org/10.1145/1281100.1281118

[20] V. Setty, M. van Steen, R. Vitenberg, S. Voulgaris, Poldercast: Fast, robust, and scalable architecture for p2p topic-based pub/sub, in: P. Narasimhan, P. Triantafillou (Eds.), Middleware 2012, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 271–291.

[21] G. Chockler, R. Melamed, Y. Tock, R. Vitenberg, Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication, in: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, DEBS '07, Association for Computing Machinery, New York, NY, USA, 2007, p. 1425. doi:10.1145/1266894.1266899.
URL https://doi.org/10.1145/1266894.1266899

[22] Y. Teranishi, R. Banno, T. Akiyama, Scalable and locality-aware distributed topic-based pub/sub messaging for iot, in: 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, 2015, pp. 1–7.

[23] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga, D. Kobayashi, A scalable and low-cost mqtt broker clustering system, in: 2017 2nd International Conference on Information Technology (INCIT), IEEE, 2017, pp. 1–5.

[24] S. Sen, A. Balasubramanian, A highly resilient and scalable broker architecture for iot applications, in: 2018 10th International Conference on Communication Systems & Networks (COMSNETS), IEEE, 2018, pp. 336–341.

[25] R. Banno, J. Sun, M. Fujita, S. Takeuchi, K. Shudo, Dissemination of edge-heavy data on heterogeneous mqtt brokers, in: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), IEEE, 2017, pp. 1–7.

[26] J.-H. Park, H.-S. Kim, W.-T. Kim, Dm-mqtt: An efficient mqtt based on sdn multicast for massive iot communications, Sensors 18 (9) (2018) 3071.

[27] A. Schmitt, F. Carlier, V. Renault, Dynamic bridge generation for iot data exchange via the mqtt protocol, Procedia computer science 130 (2018) 90–97.

[28] A. Schmitt, F. Carlier, V. Renault, Data exchange with the mqtt protocol: Dynamic bridge approach, in: 2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring), IEEE, 2019, pp. 1–5.

[29] T. Rausch, S. Nastic, S. Dustdar, Emma: Distributed qos-aware mqtt middleware for edge computing applications, in: 2018 IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2018, pp. 191–197.

[30] L. Stagliano, E. Longo, A. E. C. Redondi, D-MQTT: design and implementation of a pub/sub broker for distributed environments, in: 2021 IEEE COINS: IEEE International Conference on Omni-layer Intelligent systems (COINS), IEEE, 2021.

[31] R. Banno, J. Sun, S. Takeuchi, K. Shudo, Interworking layer of distributed MQTT brokers, IEICE Transactions on Information and Systems E102.D (12) (2019) 2281–2294. doi:10.1587/transinf.2019pak0001.
URL https://doi.org/10.1587/transinf.2019pak0001

[32] F. Rahimian, S. Girdzijauskas, A. H. Payberah, S. Haridi, Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks, in: 2011 IEEE International Parallel Distributed Processing Symposium, 2011, pp. 746–757. doi:10.1109/IPDPS.2011.75.

[33] E. Longo, A. E. Redondi, M. Cesana, P. Manzoni, Border: a benchmarking framework for distributed mqtt brokers, IEEE Internet of Things Journal (2022) 1–1doi:10.1109/JIOT.2022.3155872.